WL-TR-91-3032

(2)

ADVANCED LAUNCH SYSTEM (ALS) SPACE TRANSPORTATION EXPERT SYSTEM STUDY

Gerry Szatkowski

General Dynamics Corp.
PO Box 85990
San Diego CA 92138

March 1991

Final Report for Period October 1987 - March 1990

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6553

NOTICE

WHEN GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA ARE USED FOR ANY PURPOSE OTHER THAN IN CONNECTION WITH A DEFINITELY GOVERNMENT-RELATED PROCUREMENT, THE UNITED STATES GOVERNMENT INCURS NO RESPONSIBILITY OR ANY OBLIGATION WHATSOEVER. THE FACT THAT THE GOVERNMENT MAY HAVE FORMULATED OR IN ANY WAY SUPPLIED THE SAID DRAWINGS, SPECIFICATIONS, OR OTHER DATA, IS NOT TO BE REGARDED BY IMPLICATION, OR OTHERWISE IN ANY MANNER CONSTRUED, AS LICENSING THE HOLDER, OR ANY OTHER PERSON OR CORPORATION; OR AS CONVEYING ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY IN ANY WAY BE RELATED THERETO.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

JOHN B. SCHROEDER
CONTROL ELECTRONICS DEVELOPMENT ENGR

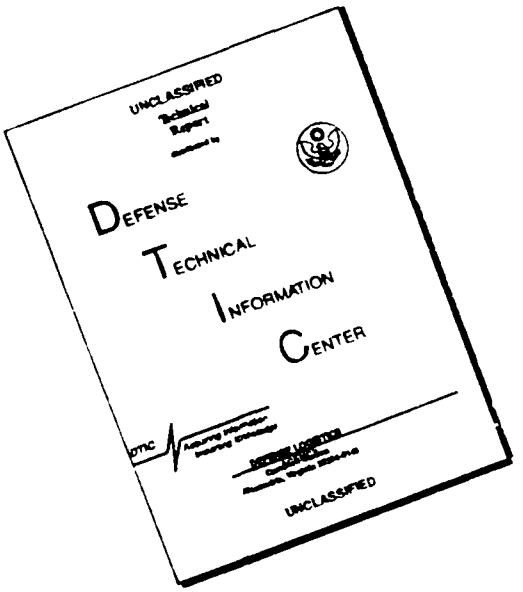
GREGORY J CECERE
CONTROL SYSTEMS DEVELOPMENT BRANCH

DAVID K. BOWSER
ASST CHIEF
FLIGHT CONTROL DIVISION

IF YOUR ADDRESS HAS CHANGED, IF YOU WISH TO BE REMOVED FROM OUR MAILING LIST, OR IF THE ADDRESSEE IS NO LONGER EMPLOYED BY YOUR ORGANIZATION PLEASE NOTIFY WL/FIGL, , WRIGHT-PATTERSON AFB, OH 45433- 6553 TO HELP MAINTAIN A CURRENT MAILING LIST.

COPIES OF THIS REPORT SHOULD NOT BE RETURNED UNLESS RETURN IS REQUIRED BY SECURITY CONSIDERATIONS, CONTRACTUAL OBLIGATIONS, OR NOTICE ON A SPECIFIC DOCUMENT.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) WL-TR-91-3032
6a. NAME OF PERFORMING ORGANIZATION General Dynamics Corp. 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Flight Dynamics Directorate (WL/FIGL) Wright Laboratory
6c. ADDRESS (City, State, and ZIP Code) PO Box 85990 San Diego CA 92138	7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6553
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-87-C-3620
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
63224C	2403	07	25

11. TITLE (Include Security Classification)

Advanced Launch System (ALS) Space Transportation Expert System Study

12. PERSONAL AUTHOR(S)

Gerry Szatkowski, General Dynamics

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Oct 87 TO Mar 90	March 1991	275

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
22	04		
23	08		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Advanced Launch System (ALS) Space Transportation Expert System Study examines the issues of incorporating knowledge-based technology into the ALS program. The Study results provide: Methodologies for recognizing when the technology is applicable, a standardized architecture for incorporating knowledge-based systems, and a methodology for validation and verification of systems developed with this technology.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
J. B. Schroeder	513-255-8431	WL/FIGL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

Table of Contents

Acknowledgments

This study was done under the auspice of the Wright research and Development Center, (WRDC) WPAFB, Oh. It represents 4 man-years of effort from Oct. '87 thru Mar. '90.

The WRDC Task Managers were:	J.B. Schroeder,	PH: (513) 255-8431
	Gary Smith,	PH: (513) 255-8432
The government review committee was:	Douglas Price	LaRC, NASA
	Maj. Mike Taint	Space Div., USAF, LA,Ca.
	Ray Bortner	WRDC, WPAFB,Oh.
	James Starr	Aerospace Corp. LA,Ca.

The following people have contributed to this report:

GENERAL DYNAMICS
SPACE SYSTEMS DIVISION

Dr. Gerard Szatkowski	**Project Manager, (619) 496-7093**
Angelique Crane	Technical review
Barry Levin	Technical Lead: Candidate Systems, Autonomous Arch.
Carlin Lowry	Candidate Risk Assessment, Risk / Performance Matrices

SUBCONTRACTOR
ABACUS PROGRAMMING CORPORATION

Dr. Roger Schultz	**Technical Lead**: K-Bus Concepts
Malcom Currie	Contributor to autonomous architectures concepts
James Geissman	Contributor to V&V concepts
Charles Hartmann	Contributor to autonomous architectures concepts
Martin Hyman	Contributor to autonomous architectures concepts
Iain Stobie	Assessments Methodologies, and K-Bus Concepts
Loren Meck	Contributor to V&V concepts

1. INTRODUCTION

The Advanced Launch System (ALS) program, a joint DoD/NASA effort, is an aggressive attempt to realize much lower costs and turnaround times with improved safety. Meeting the goals of the program will require fundamental changes in launch concepts, including much higher levels of automation and autonomy. Increasing autonomy will require the incorporation of techniques from artificial intelligence, such as knowledge-based systems, in order to replace what has traditionally been a human role.

The advanced development program (ADP) study reported in this document examined the issues in incorporating this knowledge-based technology into the ALS program. ***The results represent a major contribution to providing the technology necessary to realize the ALS goals.*** *These results, discussed below provide methodologies for recognizing when the technology is applicable, provides a standardized architecture for incorporating knowledge-based systems with conventional systems, and provides a methodology for validation and verification of systems developed with this technology.*

This report presents the results of research into Expert Decision Aid Systems, ADP 2301. The research was sponsored by the Writght Research & Development Center at WPAFB, Oh. and conducted by General Dynamics Space Systems Division with the assistance of the Abacus Programming Corporation. It has the following sections:

Section 1: Introduction
Section 2: Expert System application assessment
Section 3: Guidelines for achieving maximum autonomy in ALS
Section 4: The Knowledge-Bus architecture
Section 5: Verification & Validation of Knowledge-Based Systems

- Section 1 is an introduction and summary of the entire report.
- Section 2 presents a new and innovative methodology for ranking expert system applications in ALS or similar programs. The methodology is applied to the area of ALS launch vehicle and associated ground systems. A number of candidate applications are identified and ranked.
- Section 3 presents an analysis of the maximal feasible extent autonomy in ALS operations, and how the use of knowledge-based systems can help increase autonomy. A design approach to this degree of autonomy will be demonstrated in Phase 2 (ADP 2302).
- Section 4 presents a software architecture for ALS, where a number of cooperating, distributed knowledge-based and procedural systems will be developed and evolve with technology upgrades during the long anticipated life of the program. System-level components of the architecture will be developed and demonstrated in Phase 2 (ADP 2302).
- Section 5 presents the framework for a methodology to achieve verification and validation (V&V) of expert or knowledge-based systems in ALS-type applications. A specific methodology will be applied in Phase 2 (ADP 2302).

1.1 SCOPE

This report gives the results of analyses of issues arising from selected and effective use of expert or knowledge-based systems (i.e., artificial intelligence techniques) in ALS. Because it is a Phase 1 ADP research report, there is no

formal identification of CPCIs, no code, and none of the other products required by DoD-STD-2167A, even though some software design is included.

The research project underlying this report began (kickoff meeting) on November 20, 1987. The first task was the development of the expert system application assessment methodology and researching AI development architectures. A preliminary version of the expert system application assessment methodology was completed in February, 1988. The project then experienced a hiatus from February 19, 1988 until October 12, 1988.

When the project resumed, the tasks centered on (a) the assessment methodology and (b) AI development architectures, with the latter thread eventually leading to a distributed knowledge-based system architecture called the Knowledge Bus. From an analysis of launch vehicle systems and operations, 33 candidate applications were identified to test the methodology. The methodology was extended, converted to tabular form suitable for spreadsheet implementation, applied to the 33 candidate systems, and documented in Section 2. A feature analysis of distributed and real-time knowledge-based systems, based on a literature review and hands-on examination, served as a foundation for the Knowledge-Bus architecture, which was completed and documented in Section 4.

Concurrent with the ALS Phase II pre-design activity, the analyses underlying Section 3 (maximum autonomous architecture) and Section 5 (Verification & Validation (V&V)) received priority. All tasks have now been completed in this edition.

1.2 OVERVIEW

The major objective of thus project is to maximizing autonomous operation with focused attention on the use of artificial intelligence (AI) methods. Effective use of AI (or knowledge-based systems—KBS) requires selection of appropriate applications, an appropriate development (including V&V), and an operations framework. This report:

- describes a methodology for evaluating applications for KBS implementation, and a set of appropriate applications within the launch vehicle domain, ranked according to payoff when compared to a conventional implementation
- analyzes the implications of autonomy and derives a distributed architecture employing a combination of KBSs and procedural programs
- describes an implementation concept for this architecture (the 'Knowledge-bus') that provides the framework for *distributed and cooperating* systems, including KBSs, procedural applications and support systems such as databases, sensors and effectors
- gives guidelines for the V&V of KBSs built under this framework

1.3 LINEAGE

The research described in this report was developed to fit within the framework of the overall ALS program. The ALS System Requirements Document (SRD) and the ALS Technical Reference Document (TRD) specify general requirements to use artificial intelligence (AI) and concept-encapsulation (refered to as object-oriented programming) techniques, minimize human judgment in data-intensive time-critical applications, and to reduce the use of complex equipment and labor intensive operations.

Our STAS studies, based on Space Shuttle and Atlas Centaur, have shown that currently the major recurring launch vehicle costs break down as follows:

- Logistics & ground segment: 46.8%
- Design, development, test & evaluation: 15.4%
- Facilities & ground equipment: 11.0%
- Reusable hardware: 26.8%

The first three total 73.2% of the recurring cost, and are relatively amenable to reduction through: increased automation (spreading automation to more tasks, improving integration, reducing manpower costs), improved software (improving speed, accuracy and reliability), and increased productivity in software development and maintenance (fewer staff producing better software, sooner and with fewer errors). A 30% reduction in these three

areas—which is a reasonable target for the technology described in this report—would represent a 22% reduction in the overall recurring cost. Figure 1.1 summarizes these costs and the effect of a 30% reduction.

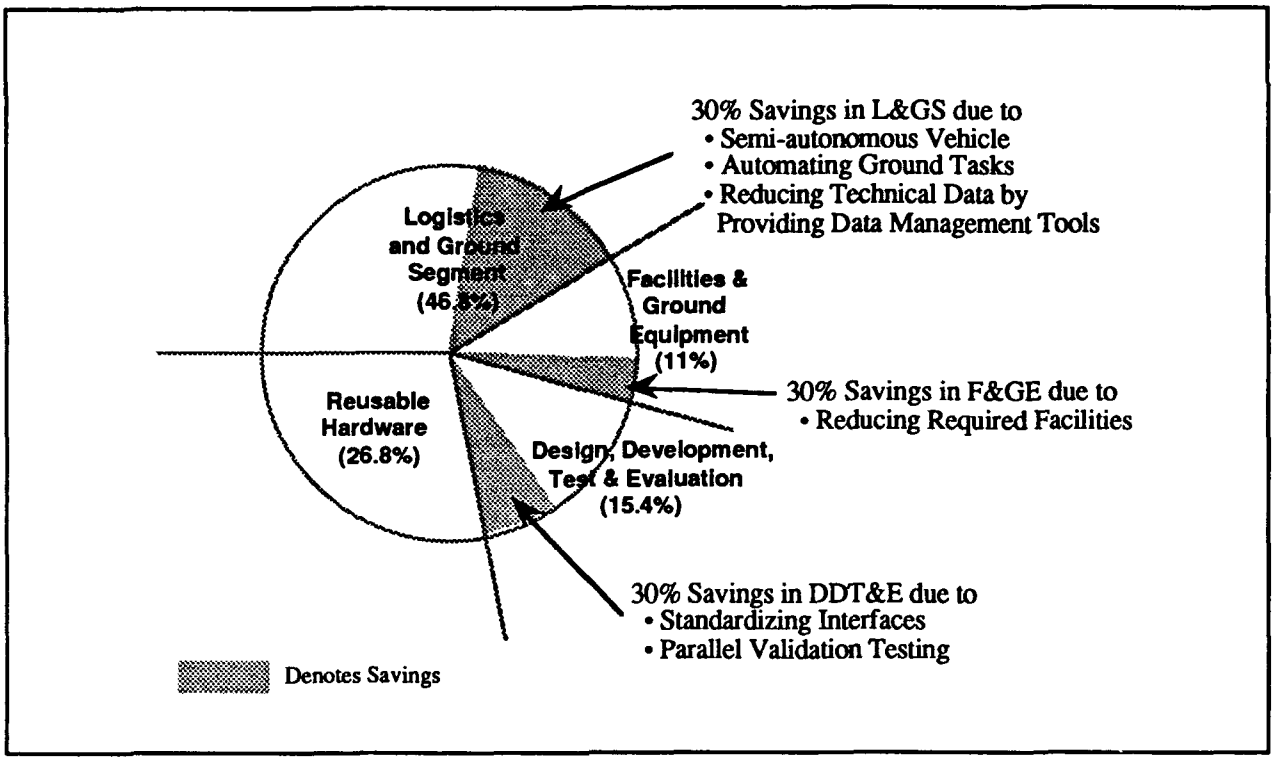


Figure 1.1—Increased automation could yield a 30% reduction in cost

With this as background, our extensive domain knowledge of launch vehicle operations was combined with the innovative methodology for identifying expert or knowledge-based system candidates, to produce the rankings in Section 2. Section 3, the analysis of maximum autonomy, derives from the SRD and TRD, the autonomy requirements (which are substantial), plus defines the set of selected applications, and gives insights into the nature of knowledge-based systems to be used.

The implementation concept (described further as the Knowledge-bus or 'K-bus') architecture for cooperating distributed systems in Section 4 derives from:

- the maximum autonomy analysis,
- specified UNIS goals (database access, data management, application toolset) in the SRD and TRD,
- ALS' intended long lifetime, and
- the set of selected applications.

Section 5 discusses in detail aspects of V&V for knowledge-based systems, predicated on the The K-Bus architecture, the set of specific applications, and implementation methodology considerations. These interactions of these threads are summarized in Figure 1.2.

1.4 ACRONYMS AND ABBREVIATIONS

The following acronyms and abbreviations are used in this document:

AGN&C	adaptive guidance, navigation and control
AI	artificial intelligence
ALS	Advanced Launch System
ALSYM	ALS simulation system
DE	domain engineer

CASE	computer-aided software engineering
COTS	commercial off-the-shelf (software)
ES	expert system
GSE	ground support equipment
KBS	knowledge-based system (expert systems, model based reasoning, etc...)
K-Bus	a software architecture for cooperating distributed knowledge systems
KE	knowledge engineer
MPRAS	*multi-path redundant avionics suite*
OS	operating system
SRD	ALS System Requirements Document
STAS	Space Transportation Architecture Studies project
TRD	ALS Technical Reference Document
UNIS	ALS Unified Information System
V&V	verification and validation

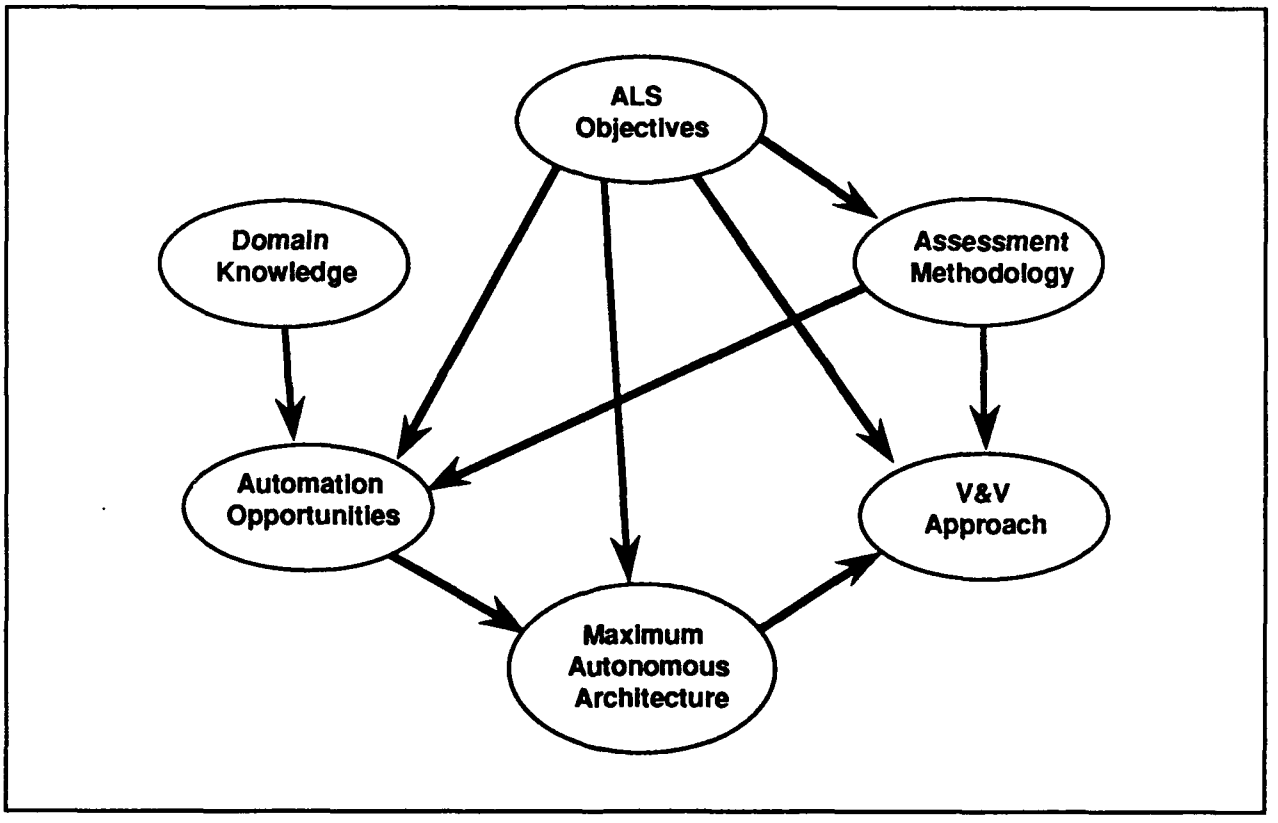


Figure 1.2—This study follows from ALS objectives

1.5 EXECUTIVE SUMMARY

This section summarizes the major arguments and analyses of Sections 2 through 5. In this summary complimentary treatments have been consolidated and ALS program goals are explicitly considered.

1.5.1 ALS Objectives

The overall objectives of ALS, although high-level and broad, can be used to derive some measurable/realizable goals, which in turn can be used to derive specific automation tasks. Instead of anticipating radical technological

breakthroughs, attention is focused on streamlining and reducing the cost of recurring processes. From this can be derived specific and realistic goals in areas such as:

- reducing the cost of replacement, maintenance and redesign of ALS components and support systems over the long operational lifetime of ALS
- linking constituent ALS systems in a common framework to ease maintenance and enhance cooperation or synergy
- reducing the cost of using automated systems
- reducing the cost of developing and maintaining the many automated systems that will be employed through productivity-enhancing methodologies and the reuse of modules (including COTS (commercial off-the-shelf software))
- maximizing autonomous operation and/or minimizing reliance on human judgment, including during the final hours prior to launch (Section 3 addresses this in more detail)

Specific automation tasks can in turn be derived from these and similar goals, such as those described in the ALS Technical Reference Document. This report, then, is concerned with the technology that will be required to implement the specific automation tasks and thereby realize the goals and overall mission objectives.

1.5.2 Derived automation goals and tasks

Specific automation goals will follow from the general goals in support of ALS objectives. The following paragraphs review (a) some of the goals that have strong software implications and (b) some specific software development tasks derived from the goals. Simply mandating good practices (e.g., design standards) does not ensure that real progress is made toward ALS goals. What is required is *toolsets and methods* that make it easier to do things right in the first place, and many of the tasks involve the creation of tools. The body of the report considers specific instances of these tasks (Section 2), a software infrastructure (Section 4), and a methodology for verifying KBSs implemented as a result (Section 5).

Goal: Reduce (software) development/replacement/maintenance costs. The following are some specific automation tasks that support this goal:

- define standard software interfaces to permit the development of plug-compatible components
- develop a software architecture of standard and reusable components that are implemented at a high level of abstraction ("generic applications") rather than merely parameterized modules that encapsulate a few lines of code
- develop software in object-oriented languages to permit the architecture of reusable components to have a common definitional basis
- build verification and validation support into software components

Goal: Maximize synergy between subsystems/subactivities. The following are some specific automation tasks that support this goal:

- maximize the reuse of data/information (only collect it once, as early as possible), e.g., flow data from CAD/CAM to all subsequent manufacturing stages and derive tests from it automatically.
- develop automation applications from top-down analyses using libraries of common abstractions and high-level concepts (e.g., hierarchical design by planned selection and refinement underlies many activities including configuring payloads and planning mission sequences) rather than bottom-up case-by-case, this maximizes inter-application compatibility

Goal: Enhance system usability and improve user interfaces. The following are some specific automation tasks that support this goal:

- provide a library of high-level interface components (e.g., menus, widgets, buttons, *a la* Macintosh or Windows) to enable developers to produce consistent and intelligent user-software interfaces

- provide a library of consistent and intelligent software-to-software interface functions to assist the development of networks of cooperating systems
- provide a library of consistent and intelligent hardware-to-software interface functions to assist the development of networks of cooperating systems

Goal: Minimize need for human judgment, i.e. maximize autonomy. The following are some specific automation tasks that support this goal:

- include the basic elements of knowledge-based systems or expert systems, such as inference engines and knowledge encapsulation to perform basic reasoning as part of the software architecture
- include built-in support for distributed, cooperating yet independent modules in the software architecture
- include support for fault-tolerance in the software architecture
- develop tools to help optimize compatibility of procedural and knowledge-based systems to implement autonomous tasks

The common theme that emerges here, from which other specifics are derived, is a stress on autonomy, automation and maintainability. Earlier launch systems (Atlas, Titan, Space Shuttle, etc.) have islands of automation, but because of a lack of end-to-end automation and autonomy, the expense is high, delays are too common, and the "standing army" is still required. (It also can be hard to separate marginal costs of payloads from booster costs, which may make boosters seem to be more expensive than they are.) The technology described in this report specifically addresses issues of autonomy and maintainability.

1.5.3 Role of Knowledge-Based Systems — Assessment Methodology (Section 2)

ALS requires operational effective and cost beneficial automation. The questions are: *what* to automate, and *how* to automate it. Further inspection asks: where to subdivide into subsystems, and whether to use procedural or knowledge-based software implementations? An implementation evaluation methodology (knowledge-based or procedural?) belongs in the system design process as an aid to answering these questions.

Many automated systems will be required to achieve the autonomy goals, and a major design task will be to partition the entire ALS system flow into individual automation systems and determine whether their software should be implemented as procedural or knowledge-based systems. Section 2 identifies a select thirty-three candidate automation applications, grouped under decision aid areas in: health monitoring, pre/post launch analysis, and mission planning and analysis. Each of these may be a single system or a set of cooperating systems. Although a different partitioning could result in a different set of candidates, the thirty-three cover a wide range and effectively represent the spectrum of automation and autonomy opportunities within ALS, and provide an effective demonstration of the evaluation methodology.

Given candidates for automation, considerations such as risk avoidance and implementation efficiency dictate that the decision of knowledge-based or procedural implementation be made on a rational basis, considering the potential risks and benefits of the different techniques and available development resources. Section 2 presents an objective, repeatable methodology for evaluating candidate knowledge-based applications.

The assessment methodology considers systems in terms of specific criteria related to factors of KBS suitability, implementation cost, operational cost, implementation risk and operational risk. The data is combined to yield the following integrated (output) attributes:

- operational cost improvement
- turnaround reduction
- flight safety improvement

KEY BENEFITS: (Section 2)

1) General purpose assessment methodology for KBSs

2) Relative ranking of ALS AI candidate applications

OBJECTIVE:

- **Assess benefits & risks of candidate applications**
- **Be consistent, accountable, automatable**

RESULTS:

- **Obtained candidates from opportunity study**
- **Integrated 17 unit attributes for each candidate**
- **Applied implementation-specific weights**
- **Revised on basis of cost, turnaround, safety improvements**
- **Scaled and ranked according to weighted merits**

	CANDIDATE SYSTEM	Final Assessment
22	Mission Planning with Automated Navig	38 49
15	Pre-Flight Test Analysis	32 00
26	System Wide Event Correlation	30 38
28	Facilities Manager	29 34
29	Mission Design Automation	29 27
16	Post Flight Telemetry Data Analysis	28 72
● 27	Vehicle Test Conductor/Scheduler	27 15
24	Command and Control Scheduler	26 78
31	Payload Manifesting	23 81
11	Vehicle Processing Logger System	23 05
10	Operation Troubleshooting	22 15
● 9	Launch Complex Envron Control System	19 48
6	Integrated Test Ctlr for Vehicle System	19 27
5	Operator Training Simulator	19 14
● 19	Propellant Tanking of Vehicle	18 13
18	Pneumatics, Press., and Purge Controls	17 03
25	Support for the Decision to Launch	16 79
32	Countdown Operations System Monitor	16 77
23	Range Safety System	16 32
8	Automatic Recorder Assignment	16 05
3	Critical Parameter Vehicle Surveillance	15 61
21	Guidance Calibration	15 25
2	Limit Testing	14 85
33	Telemetry/Landlines Checks & Assignm	14 57
17	Flight Control Power Applic. and Monito	13 78
30	Range Safety Sys. and Recovery Operatic	13 55
7	Automatic Remote Sensor Calibration	13 37
4	Hazardous Gas Identification and Safetm	12 61
20	Engine Ignition Ground Perform Mon	11 75
12	In Flight Engine Perform Monitor	8 03
13	Fluids Analysis Health Monitoring	7 83
14	Abort/Alternative Mission Modes (AGN&	6 41
1	Data Compression Analysis	5 58

Figure 1.3—Final Relative AI Candidate Ranking

1.5.4 Maximum Feasible Autonomy (Section 3)

Identification of initial knowledge-based system opportunities permits the construction of KBSs at appropriate points. The next step is to consider the general question of maximum autonomy for an overall end-to-end ALS system design, and whether there are basic design principles that might alter the balance of factors to permit even greater autonomy than envisaged in the isolated KBS evaluation methodology. The extent of automation can potentially be extended farther in the gaps between these isolated automation points is a function of the design being able to integrate knowledge-based and procedural systems over a distributed network.

Key to this analysis is a series of studies to define the maximum feasible extent of autonomy in systems like ALS. First, we assessed the advantages in terms of flexibility, technology transparency, and ease of V&V vs. the risks of development cost unknowns, and safety concerns. Next are vehicle-ground tradeoffs involving advanced complimentary technologies (made possible by increased computing power), specifically, advanced avionics, and adaptive systems all which may affect the architecture and design principles. This maximum autonomy analysis is presented in Section 3. An example architecture for maximum autonomy is shown in Figure 1.4.

KEY BENEFITS: (Section 3)

1) A philosophy supporting a distributed, flexible architecture linking AI and procedural software

2) Tradeoffs criteria for Vehicle/Ground automation

OBJECTIVE:

- **Assess advantages for an integrated automation approach**
- **Develop vehicle-ground automation tradeoff criteria**

RESULTS:

- **Performed architecture trades analyses:**
  - **Expert Systems vs. Conventional Programming**
  - **Allocation of functions On-board vs. Ground**
  - **Distributed vs. Centralized Control**
  - **Object Oriented concept encapsulation vs. Modular S/W**
- **Defined vehicle/ground tradeoff criteria for launch systems**
  - **Major sub-tasks include: Planning**
    Monitoring
    Diagnostics
    Configuration Cntl. /Training
- **Defined major features required for a linking architecture**
  - **Communications (form & function between control entities)**
  - **Configuration/Control Management**
  - **Use of COTS (Commercial of the Shelf Software)**
  - **Use of hardware and software standards**

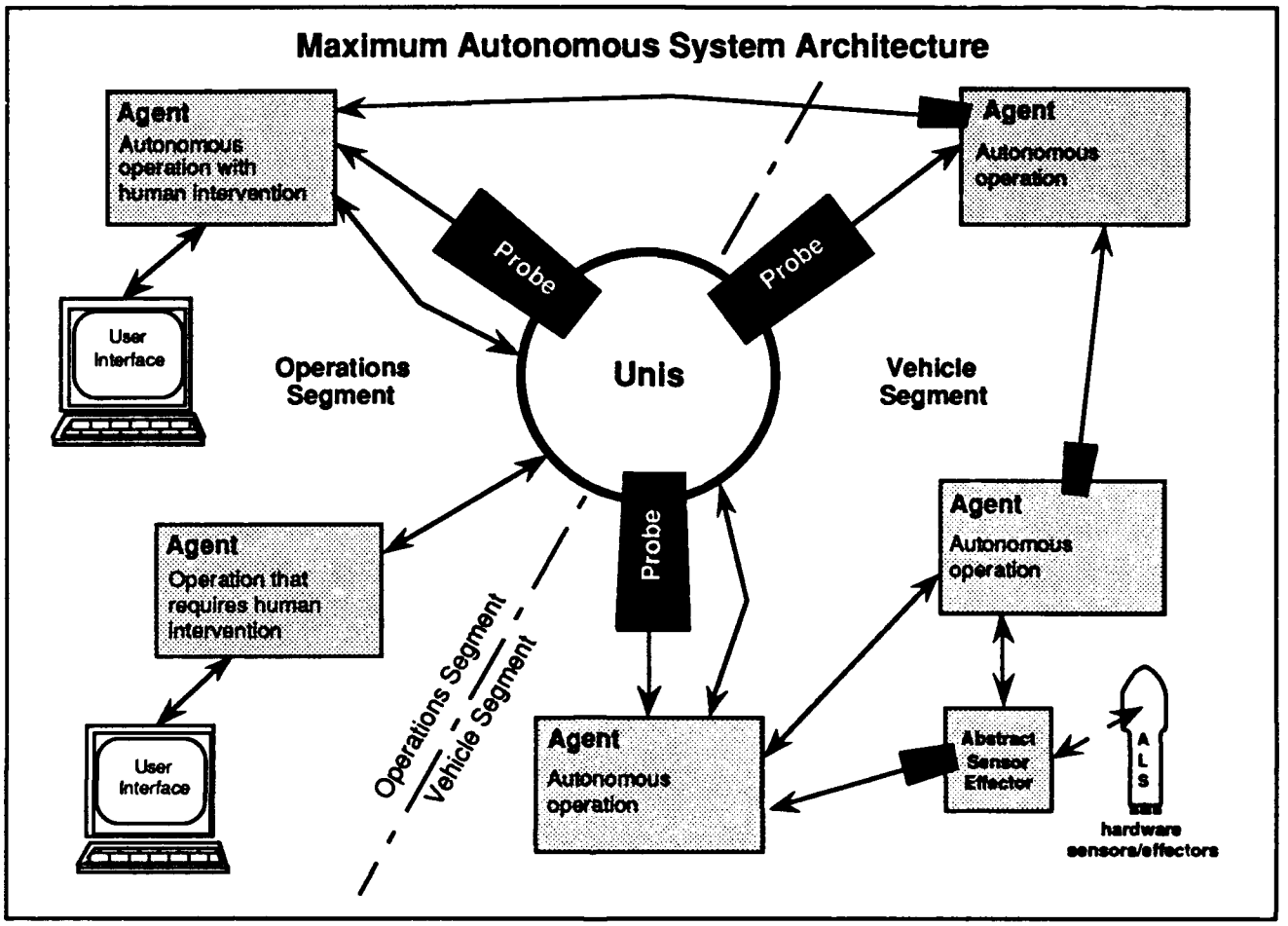


Figure 1.4—Distributed Control Approach to Max. Autonomous Architecture

1.5.5 Automation Architecture for Autonomy (Section 4)

This section documents the salient features required to implement just such an end-to-end architecture, with considerations of KBS feasibility, distributed control autonomy, and V&V. Current launch vehicle systems employ basicly a single computer program to perform check/out and launch operations. By judiciously dividing the tasks into appropriate subsystems and implementing each in flexible environments, it may be operationally and cost effective to implement the maximum feasible autonomy identified in Section 3. However, building a system consisting of a number of distributed, cooperating elements combining procedural and knowledge-based programs with support systems such as databases, graphics engines, etc., is a major challenge, even with today's technology. Today, merely getting two intelligent subsystems to cooperate can be difficult, not to mention an entire network as with ALS. What is needed to facilitate this degree of cooperation is a system-level architecture that includes the necessary facilities as a sort of extended operating system. One that is able to freely go from machine to machine, operating system to operating system. The Knowledge-Bus (K-Bus) information-passing protocol is just such an architecture to layer on top of conventional 1990's hardware for real-time control and non-R/T support communications.

Section 4 describes the K-Bus, a layered architecture leading to an object-oriented programming (OOP) implementation of distributed cooperating system. OOP differs from previous design methodologies in that it considers the operating environment as being made up of objects that can both take actions and have actions taken against them. Another major difference is that the data structures, functions, and procedures that exist in other designs are inherently part of an object here.

The architecture provides guiding implementation principles and structures that will be applicable throughout the lifetime of ALS. The features to be implemented in the K-Bus will provide underlying tools for ease of development

and the coordinating functions that permit diverse applications (knowledge-based and procedural) to operate as a coherent system. The underlying drivers of the K-Bus concept are depicted in Figure 1.5.

The layers of the K-Bus architecture are shown in Figure 1.6.

KEY BENEFITS: (Section 4)

1) **Specification for a distributed cooperative control architecture incorporating intelligent communication between knowledge-based elements ('Knowledge-bus')**
2) **Key requirements for commercial-off-the-shelf (COTS) software to comprise K-bus features.**

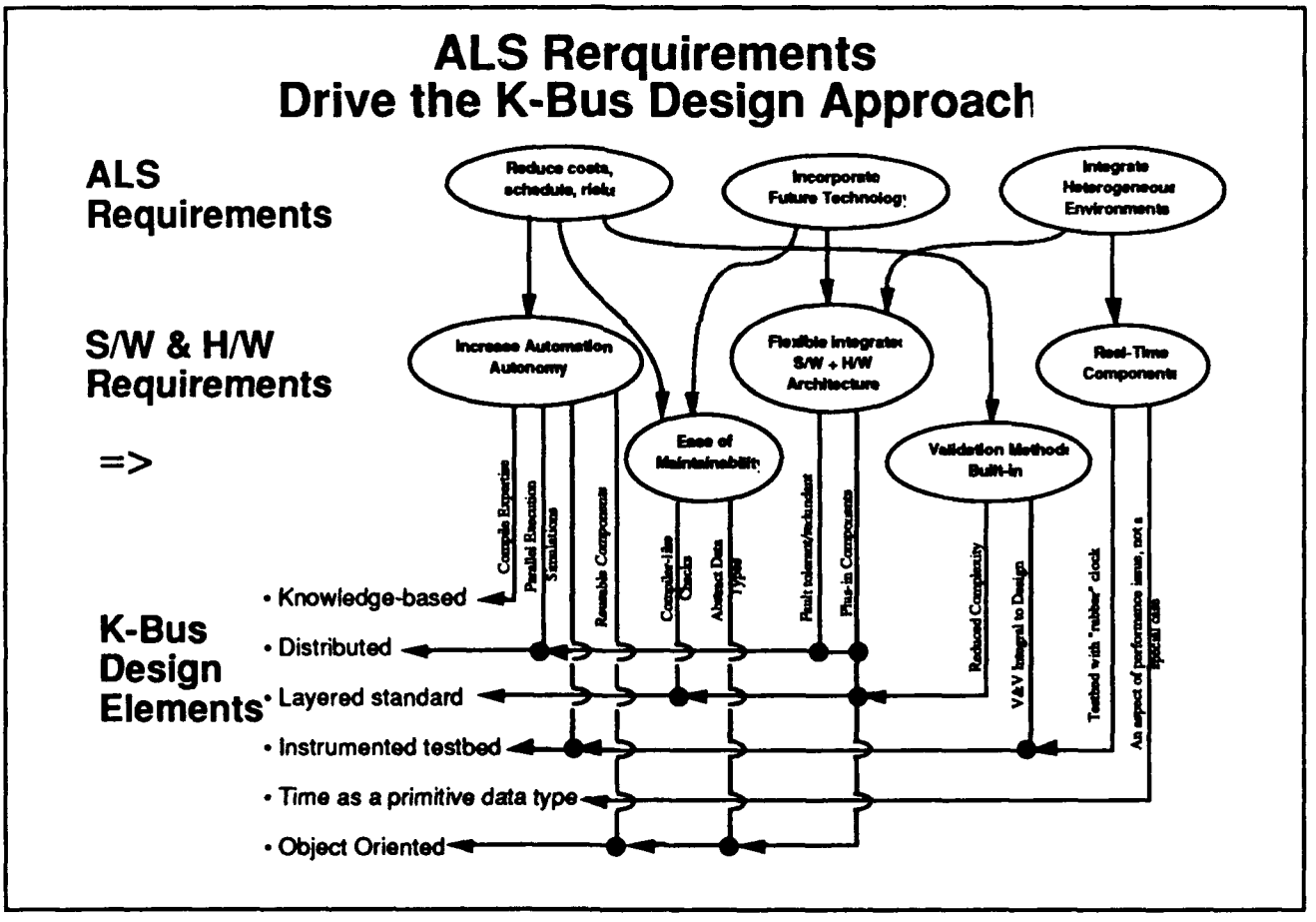


Figure 1.5—Origin of the K-Bus Concept

1.5.6 Verification & Validation of Autonomous KBSs (Section 5)

If knowledge-based systems are to have a any role supporting real-time control, highly safety critical systems, they will have to be follow a rigorous development methodology using reliable techniques for specification, design, coding and verification/validation. The first three of these can use existing, proven methods for producing high quality software. The verification/validation of KBSs, however, has been troublesomely vague and elusive with current

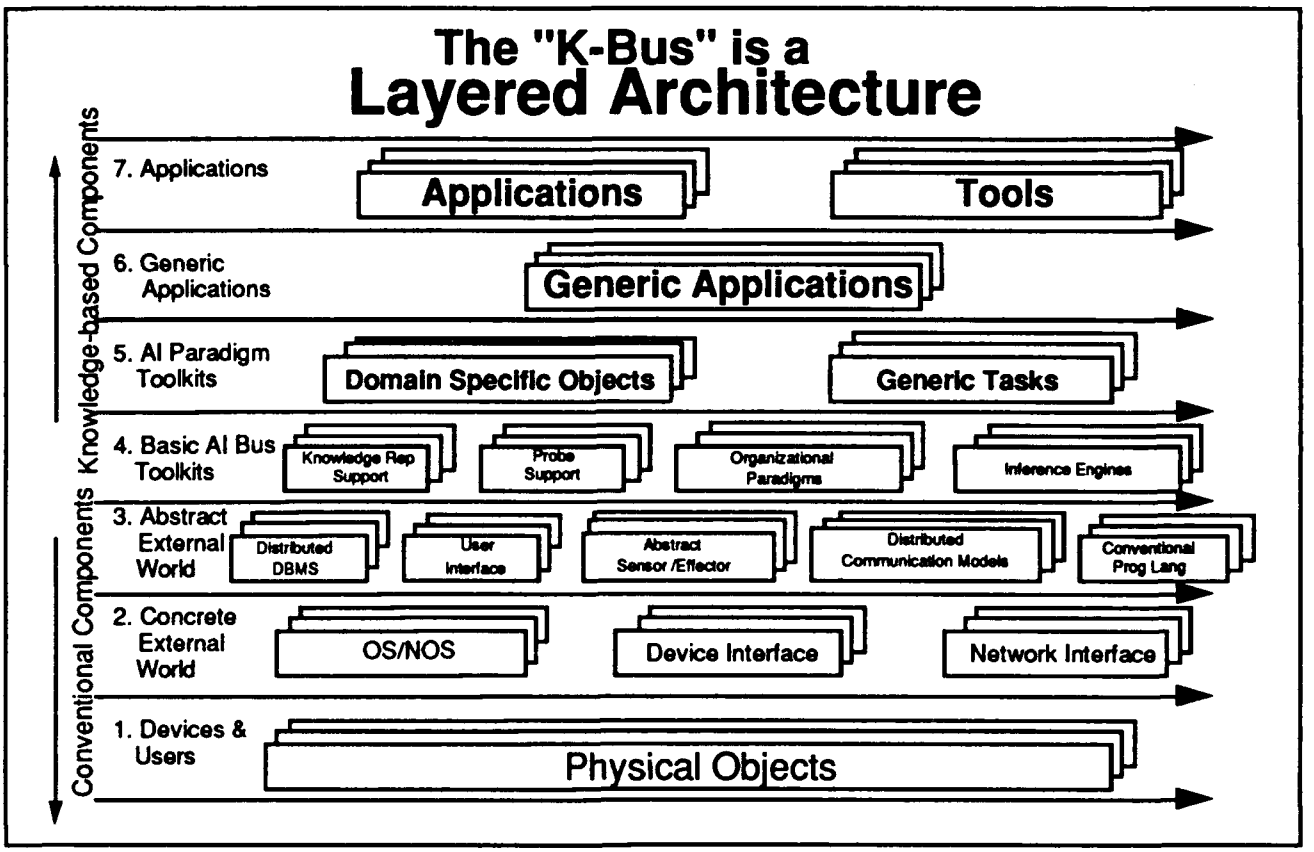


Figure 1.6—Overview of K-Bus Layers

techniques. Software verification is the process of evaluating the product of a given phase of the software development cycle to ensure correctness and consistency with respect to the products and standards provided as input to that phase. Validation on the other hand is affirmation, i.e. an evaluation of a system to ensure compliance with specified requirements; the code can be said to correctly implement the intended requirements. Simply put — *verify* that the software does what the designer programmed it to do, and *validate* that the system does what the user wanted in the first place.

Section Five offers a sufficient V&V methodology for knowledge-based systems for highly critical systems. It discusses specifications, development methodology, and automated tools to assist verification (see Figure 1.7). The major points of the V&V methodology are summarized in the following points:

- *Maximizing validity* of systems by minimizing the gap between requirements and implementation, resulting in better (i.e., more likely to be satisfied and more likely to be useful) requirements and implementations. Not only does this enhance validity, it simplifies validation.
- *Tailoring the level of requirements* through

  a) top-down end-to-end analysis of the total system,

  b) using generic tasks and applications as model requirements, and/or

  c) using prototypes to generate requirements where the problem is not well-defined, one of replacing a human expert, or integrating lower-level systems.

- *Raising the level of implementation* through the development of a library of generic applications, based on generic tasks including knowledge processing heuristics, basic knowledge and commonly-used modules such as standardized user interface components
- *Using prototypes to explore design spaces* where an implementation cannot be specified as a combination of library components or other known techniques such as mathematical models or procedural programming structures
- *Simplifying and improving verification* with the use of automated tools for passive and active testing, which should be built specifically to interface with the libraries of generic components

KEY BENEFITS: (Section 5)

1) Specification of a verification/validation methodology suited to knowledge-based systems.

2) Identifies V&V techniques and tool-kits required to produce highly reliable software for critical tasks.

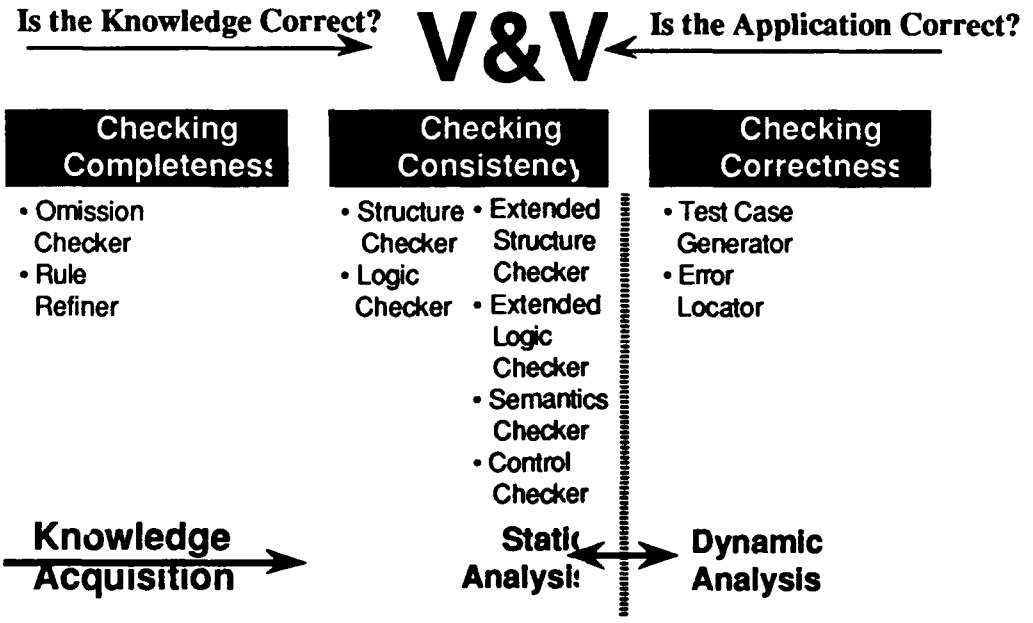


Figure 1.7—A sufficient list of V&V tool-kits required

2. ASSESSMENT

2.1 REVISE KEY DRIVERS FOR ALS BASELINE SYSTEMS

2.1.1 Areas of Functionality

In order to prove that the technology associated with expert systems is viable when integrated into the Advanced Launch System generation of space vehicles it is necessary to establish a key driver baseline. The expert system key drivers are a set of criteria that will be used to evaluate the feasibility of candidate expert systems.

The key drivers cover essentially the following four areas of functionality:

- cost
- schedule
- mission success
- safety

2.1.2 Conventional Methodology

The conventional methodology to support these key drivers are identified in this section. Knowledge of the conventional key drivers is required to identify where we are coming from. Identification of the key drivers will allow a nodal point in which we can reference expert system key driver applications.

2.1.2.1 Conventional Cost Drivers

Cost related drivers are heavily dictated by the design of the flight vehicle and related ground service equipment. The degree of design maturity plays a major weighting on the design cost driver. Older designs typically have established a refinement to achieve optimal throughput of the design functionality and tend to be more cost efficient. Designs that allow for highly autonomous operation will affect costs by both a higher vehicle cost for highly complex systems and lower servicing costs from a smaller standing army. Other factors such as design restrictions and variation in standards are governed by conditions dictated by influences outside the immediate realm of the design. A new payload requirements adding additional outside cost requirements to an additional design would be such an example.

2.1.2.2 Conventional Schedule Drivers

Schedule related drivers are driven in part by system simplicity and reconfigurability. Simplified operations, maintenance, and technology updates help maintain schedule requirements. Specifically this includes reduced maintenance, fewer repair delays, and improved system changes related to technology gains. A flexible system that is easier to reconfigure between missions will help improve mission to mission scheduling. Systems capable of performing multiple tasks or wide variations of the same task will permit shorten mission tasks schedules.

2.1.2.3 Conventional Mission Success Drivers

Drivers towards mission success are governed by performance, capability, system accuracy, robustness, and software design. Providing a system that is able to handle all reasonable contingencies will increase overall performance. This requires processors be sized with a reserve capability to handle such contingencies. A system that is capable of handling all assigned tasks and contingencies will operate more reliability and improve mission success. System accuracy will yield decisions with higher accuracy and less probability of endangering mission completion. A robust system will be less prone to failure and will insure an increased system integrity. Software design will improve mission success thru accurate, flexible, and robust operations.

2.1.2.4 Conventional Safety Drivers

Safety related drivers are governed by reliability, validation methods, redundancy. Higher reliability will directly yield higher system safety. A thorough validation methodology will detect safety related system deficiencies. Redundancy within the system hardware and software will reduce failures and increase safety.

2.1.3 Expert Decision Aid (EDA) Methodology

The expert decision aid methodology to support these key drivers are identified in this section. Knowledge of the conventional key drivers is required to identify where we are coming from. Identification of the key drivers will allow a nodal point in which we can reference expert system key driver applications.

2.1.3.1 EDA Cost Drivers

The expert system complexity, and task of application will dictate cost drivers. The components of the expert system must be designed and verified. Highly complex systems with high development cost and complex verification and validation is the primary EDA cost driver. Expert systems are effective if their task of application in which they are applied is well suited. Costs are related to the efficient assignment of expert systems to appropriate tasks.

2.1.3.2 EDA Schedule Drivers

The cope of the knowledge base is critical to definition of the schedule driver. A large and accurate knowledge base will assist in improving schedule times by providing the ability to generate all mission data quickly and accurately.

2.1.3.3 EDA Mission Success Drivers

As with conventional systems the software design will improve mission success only if it allows for design flexibility and user friendly operation. Development of the K-bus concept for standardization of the expert system linkages will allow the EDA to be flexible and portable. Available or customized user hardware can be easily integrated to the software for design flexibility. The K-bus will incorporate drivers to the system network for graphic workstation interfaces. The workstation environment will allow user friendly operation of the mission specific software.

2.1.3.4 EDA Safety Drivers

EDA safety drivers dictated by the scope of the knowledge base. An extensive knowledge base will improve safety by having knowledge that will properly react to contingencies.

2.1.4 Summary

Conventional drivers and expert system drivers are very closely related. As conventional drivers increase or decrease so do their EDA counterparts. Although these changes may be proportional they are far from equal. The magnitude of the EDA change with respect to the conventional systems will be significantly smaller.

The drivers mentioned above are sensitive to changes in the method of operation, the vehicle type, and the particular mission. As these parameters change, the requirements for real;-time verses non real-time, autonomous verses man-in-the-loop, or ground based verses airborne change, each in turn affecting the corresponding drivers for safety, mission success, cost, and schedule. Many of these changes however can be anticipated and prepared for through an adaptation in the scope of the knowledge bus.

Expert systems are capable of reviewing, assimilating, and reacting to data in quantities and at speeds far in excess of a typical human operator based system. This in turn increases the reliability and robustness of the system while reducing cost through reducing scheduling and manpower constraints. By developing an understanding of the drivers for each system as well as the elements which vary from system to system we can more accurately assess the capacity of a given expert system to maximize its potential cost reductions and system enhancements.

2.2 CANDIDATE DECISION APPLICATIONS

The expert system decision application candidates have been divided into four groups. Each of the expert system candidates have been placed into one of these four categories which are: 1.) Test and Checkout Diagnostics Aids, where there have been eleven candidates defined, 2.) Onboard Health Monitoring Functions, where there have been three candidates defined, 3.) Pre / Post Launch analysis, where there have been two candidates defined, 4.) Mission Planning and Operations, where there have been seventeen candidates defined. This section lists all of the candidates with a description, and an associated number that will be used to identify the candidate in the charts that will be listed later.

2.2.1 Test and Checkout Diagnostics Aids

2.2.1.1 Data Compression Analysis [1]

This real time data compression analysis system will accept, analyze, parse, and compress the downlink telemetry as a front end analysis routine. This will assist in accelerating the real-time telemetry data feed into the sub-system expert systems. Most of the other specialized decision systems will reside on a single multiple processor / parallel tasking computer system. The two communality points between the multiple specialist decision systems will be the telemetry feed and the common data base.

This data compression analysis decision system will allow front end decision processing on the telemetry data. This front end analysis will read the history file and look for telemetry data patterns and correlate data patterns to those that have been seen in the past. Meaningful data patterns then will be directed to the proper decision system candidate with pre-analysis footnotes attached. The effect will be to optimize the overall processing efficiency of all expert systems by reducing the amount of data any particular decision system must see. The footnotes will clue in the decision system to the telemetry data pattern and will allow bypassing the preliminary analysis within the decision system rule base. The net effect will be to eliminate the potential for redundant processing of telemetry between two decision systems.

Benefit of automation - The vectoring of data from PCM bit streams has been automated for 15 to 20 years. There is a large historical data base in which to build to create the smart rule base.

Benefit of an expert system - May help downstream expert systems

Disadvantages - The nonrecurring cost over a logic box telemetry handler is expected to be one man-year, and maintenance of 0.1 men per man-year is expected. In addition, since the vectoring of data is so crucial, it may not be realistic to assume that we will accept any risk of misdirection.

2.2.1.2 Limit Testing [2]

The real time Limit Testing Module would be located in the Launch Control Center (LCC) and test all received (COMM & PCM) filtered data against data base table limits. Data base tables must change automatically based upon events/phases and the application software being executed. Data tables provide limits for out-of-tolerance-condition and caution-level-condition testing. The limits set flags/interrupts for the operator work station, Command and Control Manager Console (CCM), and for data evaluation. Out-of-tolerance-condition's are divided into critical for launch (red-line-monitor) and not critical for launch. A red-line-monitor out-of-tolerance-condition will place the vehicle in the "Not launch ready" status. All out-of-tolerance-condition's and caution-level-condition's would be sent to the display system module for system engineering review and to the expert decision system limit handler in the work station.

This decision system is resident in the operator work station as a parallel functioning processor in the LCC computer system. This decision system will access the limit testing data base and apply an expert reasoning rule base to decipher the data for categorization and analysis. Currently with conventional means of analysis, the data base is too huge to permit full analysis. As a result the caution-level-conditions are set so tight that the operators ignore the secondary warning levels and only look at the red-line-monitor conditions. With a decision system implementation one can accelerate the analysis of the non-critical caution-level-condition to allow reliable generation of secondary warning levels.

The decision system will also access the expert data base and provide the operator the inference decisions of former expert operators. A display of recommendations will accompany the red-line-monitor display to provide the operator additional information to aid in evaluating the out-of-tolerance-condition and caution-level-condition alarms.

Benefit of automation - Today's Titan Centaur system still uses a significant number of graphics displays for human implemented redline monitoring. However, this is changing, at least on the Titan 4 program. Atlas 2 has also recently added this capability to our Computer controlled launch set, which is our version of the launch control test logic box.

Benefit of expert system - This is the first of three expert decision systems to analyze vehicle data. The other two are described in the following sections 2.2.1.3 and 2.2.1.4. When combined, the anticipated overall reduction to the launch system engineering staff is expected to be 70%, i.e., today's Atlas Centaur crew of 10 would be reduced to 3. *Note: In order to achieve the anticipated savings, it has been assumed that the vehicle mechanical and electrical operations during non-testing periods are simplified so that they are not sitting idle when tests are in progress.*

Red line monitoring can be accomplished without an expert system. However, today's redline systems have the following deficiencies. Their redline tolerances must be broader than needed to make up for the hardware tolerance of the specific hardware installed, and their limits must be reprogrammed if the configuration of the hardware changes. If these models were driven directly by the CAE and CAD data bases, the analysis would be higher in fidelity for improved quality. Improved real-time feedback of vehicle out-of-tolerance-condition and caution-level-condition alarms will increase the reliability of operator decision, and aid in training new personnel.

Disadvantage - The nonrecurring cost over a logic box red-line-monitor is expected to be one man-year, and maintenance of 0.1 men per man-year is expected.

2.2.1.3 Critical Parameter Vehicle Surveillance (Tank Watch) [3]

This real time and non-real time decision system will monitor those critical vehicle parameters that effect the static health of the vehicle when in the pre-countdown and countdown states. Extended periods of time pass when the vehicle is in this dormant state at the launch pad, or in the hanger. The intention of this decision system is to go beyond the standard redline monitor of out of tolerance conditions. This system can provide correlation of critical parameters with the support equipment performance. Should a redline value occur this decision system will provide alarm and advise the alerted operator to proper action to remedy the failure. Correlation of parameter response to support systems and the history data base will assign weighted values to aid the Vehicle Maintenance decision system, and the Environmental Control decision system to assign priorities to the equipment maintenance.

Particular emphases is to be placed on signature analysis which today's systems engineers accomplish by "knowing" what the probable cause of the signature is. Additional emphasis will be placed on using the design and analysis CAE/CAD data bases to directly generate the model base for the expert system. Note also that this system can be both real time and non-real time, since today's systems engineers operate in both modes.

Benefit of automation - The activity described above would be difficult to accomplish without either a data driven or a human staffed system. The signature of the data can be examined with fourier tools to analyze signatures, but the better approach is to use the historical time domain correlation knowledge of a systems/test engineer.

Benefit of an expert system - Besides the basic premise that this is what expert systems do best, there should be a reduced labor cost, see 2.2.1.2 above.

Disadvantage - The nonrecurring cost over a logic box red-line-monitor is expected to be one man-year, and maintenance of 0.1 men per man-year is expected. In addition proving to management that the system can be verified and will achieve the anticipated reduction in force will be difficult at first.

2.2.1.4 Hazardous Gas Identification and Safeing [4]

Provides assistance to ground crews for identification of hazardous gas leaks from the vehicle systems and from the ground support systems. Provides recommendations to ground personnel for corrective action in identification of the type of gas leak, safety egress path for evacuation of workers, and action required to isolate the leak.

Implementation of this system would require updating the current leak detection unit technology used on pad 39 for hazardous gas detection. The currently installed leak detection unit leak detector system incorporates dated technology that is maintenance intensive. The leak detection unit outputs data that requires large amounts of operator interpretation. Analysis typically reads inconsistent results. The present day unit technology is not cost justifiable and should be modernized to current technology to allow reduced costs of operation and expanded performance. The manufacturer of today's leak detection units will not produce anymore and is in the process of discontinuing maintenance support.

NASA is currently looking into replacing today's ground and airborne leak detection system at the Shuttle launch facility. NASA plans to ultimately replace the currently installed system with highly reliable mass spectrometer LDUs for the ground systems, and they are in prototype with a miniaturized mass spectrometer, developed originally for the SkyLab program, for airborne engine leak detection. This new generation of mass spectrometers are capable of yielding near real time data response

The increased performance of this new generation of leak detection units will allow much added capability to the hazardous gas identification and safeing system. These new high performance features, such as near real time data collection and simplicity for leak detection unit installation, will permit many additional leak detection units to be added to the system. This will greatly increase the amount of data which must be collected and processed by the system. Application of expert system technology will allow implementing of an inference engine rule base to apply known solutions to leak detection unit alarm conditions. The rule base will advise ground personal on zone location of hazardous leak, type of leak, recommended egress path and/or isolation procedure, and methodology to correct fault. Leak data analysis algorithms can be implemented that can analyze marginal leak situations over time and provide trending analysis. From this data a projection can be made to identify when an unsafe condition will arise.

Benefit of automation - Reduced manpower load on ground personnel to analyze leak detection unit output data, increased safety due to increased unit coverage and quicker unit response, and reduced time to check out vehicle for leak detection.

Benefit of expert system - It is possible that with the significant amount of data from ALS engines and the tank farm, which probably has significant redundancy to ensure launch availability, the data handling may be improved by expertness.

Disadvantage - Initial cost of system design and installation.

2.2.1.5 Operator Training Simulator [5]

This real time system would serve as a training platform for ground crew. Although increased vehicle autonomy on the ALS program will simplify the ground crews tasks it is advantageous to have a training simulator that will allow for better operator performance. This training simulator will simulate vehicle storage as well as launch day operations. The simulator will allow the operator to perform normal tasks in real-time to respond to both nominal and induced failure situations. The expert decision system interface will correlate from the operations history archive, previous failures and automatically induce these failures into the training simulator's operation.

Benefit of automation - This is another data driven area where, if the decision is made to automate the function, an expert approach may be best. Validates both automated and manual launch day countdown procedures.

Benefit of expert system - Increased vehicle operations reliability with reduction on possibility of operator error.

Disadvantage - The increased costs may not pay for the benefits. Especially when the operator task has been largely automated anyway, i.e. the few remaining panels are not at all complex.

2.2.1.6 Integrated Test Controller for Vehicle System Checkout [6]

This system controls integrated system tests on vehicle during vehicle checkout. It provides test results, analyzes test data, and recommends necessary corrections for the vehicle to meet specification. This system will provide for a repeatable sequential testing format. As a result all testing will be standardized from sub-system to sub-system as well as from vehicle to vehicle. This will help determine system fault tolerance.

Benefit of automation - This task is largely done today by launch control computer.

Benefit of expert system - Difficult to assess when compared to today's launch control computer logic box approach.

Disadvantage - No RIF is likely.

2.2.1.7 Automatic Remote Sensor Calibration [7]

System capable of automatic remote self calibration of vehicle and ground pressure transducers, and monitoring the manual calibration of those components that cannot be remotely calibrated because, for example, multiple remote (voting) sensors are not viable or the sensors cannot be stimulated. When in the automatic mode, the calibration system will access the manufactures' specification data base and will perform calibrations in accordance with the recommended procedure. The sensors specified tolerances will be compared to both the actual measured values and the history data base of previously measured values. The knowledge rule base will correlate the data for drift trends or failures. The rule base will alert for trends in failures within systems or sensor types and provide correction procedures and sensor performance recommendations. This system will store calibrated data in a history data base that can be correlated in a history profile algorithm. When in the manual mode, the expert system will optimize the manual calibration schedule given sensor location and calibration procedure. Results will be stored in the history data base for later analysis.

Benefit of automation - reduce manpower, refinement of calibration schedule may reduce calibration time. Note: this concept may not be applicable if a smart, self calibrating pressure transducers with built-in-test capability make it not cost effective.

Benefit of expert system - There will be up to 600 analog (pressure or temperature) transducers on ALS. If the temperature sensors remain dumb (resistance type), and if smart built-in-test pressure transducer are not available, this could save significant man-hours of calibrating/testing. (2400 per vehicle.

Disadvantage - More redundant sensors may be required than the man-hours saved are worth.

2.2.1.8 Automatic Recorder Assignment [8]

With automation of the control room it is expected that two graphic display graphics recorders will remain. This expert system will then make a determination of which parameters should be assigned to graphic recorder displays for output, and will provide self calibrating graphic scaling for the associated traces. This would allow the launch control room display recorders optimum assignment under varying situations.

Benefit of expert system - Reduces quantity of strip chart display recorders necessary for launch control center to the minimum. Reduction in both cost of equipment and number of personnel required to support system.

Disadvantage - Possible loss of data to strip chart display recorders. It is possible that there may be a scenario in which there will not be enough recorders to handle all suggested outputs. Relatively high front end cost for determining the initial requirements for simplified launch control center. It is expected to be difficult to achieve political agreement on any launch control center modification that will reduce size of standing army.

2.2.1.9 Launch Complex Environmental Control System [9]

This near real time system provides fully automated control with diagnostics for all vehicle ground equipment environmental support systems. This covers equipment related to ground services such as refrigeration, controls, hydraulics, fluid flow, etc. This decision a system will access the manufacturers' operation manual data base as well as the history data base, and will provide forecasting of potential future problems as well as active control of equipment diagnostics. Diagnostic rendering of abnormal situations shall advise the operator of equipment malfunction and related corrective action. Interface to the Vehicle Servicing and Maintenance decision system will permit coordinating preventive and required maintenance activities.

Benefit of expert system - Reduced labor costs.

Disadvantage - The cost of bringing ECS under expert control is estimated at 1 man-year. Today's complex 36 ECS system is just coming under automation, the benefits of a expert system using the KATE (Knowledge Aquisition Test Environment) shell will be evaluated using General Dynamics discretionary funds this year.

2.2.1.10 Operation Troubleshooting [10]

This system would supply analysis expertise for troubleshooting equipment failures involved with ground operations. This system would access a centralized data base containing equipment service manual information, where expertise from both the equipment manufacturer's data and ground operation personal knowledge would be stored. Retrieval of a troubleshooting procedure would be provided in a customized manner for each equipment failure.

Benefit of automation - Savings in some fault isolation diagnostics procedure retrieval.

Benefit of expert system - Knowledge capture, and systematic process would save unscheduled downtime.

Disadvantage - Systems tend to be simple and readily diagnosable. Cost of implementing an ES may outweigh benefits.

2.2.1.11 Vehicle Processing Logger System [11]

This system would test history compilation and trend analysis, and performs paper handling for checkout sequence and checklists prior to power on. This decision system tracks recurring problems and automatically inputs data to the master scheduler.

Benefit of automation - Shuttle example shows 27 hours of paper handling for every one hour spent modifying the vehicle.

Benefit of expert system - The complexity of the domain may require an expert system.

Disadvantage - The recurring costs of paper processing of QAR's and test analysis is significant in ground operations, and would be traded against LCC for automation.

2.2.2 On-Board Health Monitoring Functions

2.2.2.1 In-Flight Engine Performance Monitoring [12]

This real time decision system will be located on the vehicle (part of the vehicle avionics function). It is designed to analyze data from multiple engines, correlate the data to determine sensor validity and predict parameter levels, and give engine recommendations to respond to engine error situations. The system would also aid adaptive guidance and navigation for abort and alternate mission scenarios. To achieve this capability the decision system will evaluate a set of rules provided by the experts. The rule base will provide the flight computer recommendations such as when to shutdown an engine based on sensor fusion of abnormal limits.

Since it will be operating from ignition through the end of powered flight, it may also accomplish the task described in paragraph 2.2.1.5. Whether there would be one vehicle system or two systems (vehicle and ground) depends on the reliability allocation for this function.

Benefit of automation - Increased vehicle reliability, reduced manpower.

Benefit of expert system - If there is an anomaly in the data from the propulsion system, it is crucial to determine as fast as possible whether the problem is a sensor problem or a true engine problem. Using an expert system to evaluate all the data , with rules, should cut reaction time by at least an order of magnitude. This should increase vehicle reliability as much as 2%.

Disadvantage - None.

2.2.2.2 Fluids Analysis Health Monitoring [13]

This decision system will provide correlation of valve and sensor data in order to identify inconsistencies in the fluids system of a cryogenically propelled space vehicle. Sensor fusion analysis will allow smart decision for increased reliability.

Benefit of automation - Increased vehicle reliability, reduce manpower

Benefit of expert system - Permits better decisions to be made while performing critical fluid procedures (i.e., tanking) in the event of a failure. Also, in-flight engine performance monitoring would provide better redundancy management and thus improve mission reliability.

Disadvantage - None.

2.2.2.3 Abort / Alternative Mission Modes (AGN&C) [14]

This system would evaluate failures and engine performance variations affecting the mission trajectory. It would recommend alternative trajectory profiles to autopilot and augment adaptive guidance, navigation and controls (AGN&C). For mission abort it would plan to miss land impact trajectory.

Benefit of expert system - Improvements to mission success reliability and to mission safety, beyond a simple destruct system. Also expands launch window and thus on-time launch schedule is assured.

Disadvantage - On-board reliability concerns and increased computer processing requirements will increase costs.

2.2.3 Pre / Post Launch Analysis

2.2.3.1 Pre-flight Test Analysis [15]

This non-real time system would analyze ground test data. The test data will be correlated against a history data base of previously run similar tests, as well as against manufacturer specifications. An automatic analysis of the data will identify anomalies, out of tolerance conditions, and ther pertinent correlations. Since the analysis will correlate results against a project history file, previous testing experience will allow identification of recurring problems.

Benefit of expert system - Reduction in time and manpower. Not subject to human error; i.e., anomalies will not be overlooked. Historical correlation of test data would find and identify design flaws earlier.

Disadvantage - Initial cost to set up automation and expert system rule base.

2.2.3.2 Post Flight Telemetry Data Analysis [16]

This non-real time system will correlate the telemetry data to identify anomalies and out of tolerance conditions. The test data will be correlated against a history data base of previous mission data. The expert system rule base will be adjusted to "learn" new error conditions. Since the rule base will correlate results against a project history file, previous testing experience will allow identification of recurring problems.

Benefit of expert system - Reduction in time and manpower. Data will automatically update heuristic knowledge-base. Over 95% of the task can be routinely automated. Not subject to human error; anomalies will not be overlooked.

Disadvantage - Initial cost to set up automation and expert system rule base.

2.2.4 Mission Planning and Operations

2.2.4.1 Flight Control Power Application and Monitor [17]

This is a real-time system to checklist the application of power to the vehicle for test and preflight checkout. The system will monitor power system telemetry, and will correlate step changes with vehicle activities, logging atypical occurrences and taking action automatically to safe the vehicle in the event of a critical problem.

Benefit of automation -. Reduces normal manual monitoring and test data reduction analysis. Provides automatic safeing functions.

Benefit of expert system - Test data reduction is a complex issue when trying to correlate events to power abnormalities.

Disadvantage - None

2.2.4.2 Pneumatics, Pressurization, and Purge Controls [18]

This is a real-time expert system to automate the ground control for these systems. The system will monitor telemetry for these functions and will provide test data reduction services. Any anomalies will be identified, isolated, and safeing action initiated automatically.

Benefit of automation - Reduces normal manual monitoring and test data reduction analysis. Provides automatic safeing functions.

Benefit of expert system - Test data reduction is a complex issue for these subsystems. Fault isolation and safeing would be significantly improved, and safety would be improved by eliminating human error.

Disadvantage - None

2.2.4.3 Propellant Tanking of Vehicle [19]

This is a real time system to oversee the tanking of propellent and oxidizers onto the vehicle. System will monitor the time critical tasks of sequencing the loading of propellant on-board such as monitoring that the pressures are within range. This system will correlate the tanking parameters to vehicle history data base for the selected ALS vehicle stack configuration and payload weight. Alarms will be provided to notify ground crews of pending problems. Advanced diagnostic analysis by the expert system will advance warning time prior to redline condition. Provisions can be provided to allow this system to control the vehicle propellant loading as well as to automatically initiate corrective action to correct error conditions.

Benefit of automation - The Titan 4 is presently automating tanking.

Benefit of Expert system - LCC would be reduced due to simpler maintainable software and simpler V&V.

Disadvantage - Proving to management that the system can be verified and will achieve any benefits will be difficult.

2.2.4.4 Engine Ignition Ground Performance Monitoring [20]

This real time decision system will be located in the launch control center. It is designed to analyze data from multiple engines, correlate the data to determine sensor validities and predict parameter levels, and give engine recommendations to respond to engine error situations. To achieve this capability the decision system will evaluate a set of rules provided by the experts. The rule base will provide the ground computer recommendations such as when to shutdown an engine based on sensor fusion of abnormal limits. The performance history for new flights will be analyzed and recommendations will be provided for improvement changes to the rule base. A consolidated graphics workstation display will allow the operator a quicker understanding of abnormal situations.

Benefit of automation - This function is largely automated today.

Benefit of expert system - Possible increased vehicle reliability.

Disadvantage - Must prove system cost benefit and that verification can be accomplished.

2.2.4.5 Guidance Calibration [21]

These are computer procedures and manual tasks used by ground systems to align gyroscopes. They involve a test conductor, trend analysis, and history jacket, similar to advances in systems being developed for the shuttle.

Benefit of automation - Reduces manual data reduction and identifies potential problem earlier then would be possible otherwise.

Benefit of expert system - Improvement to operational reliability and may benefit operational costs. Also can collect heuristic data on calibration tests for trend analysis that was not possible previously.

Disadvantage - None.

2.2.4.6 Mission Planning with Automated Navigation Tailoring [22]

An automated system performs minimum tailoring of navigational parameters per mission. The system would have bounded input parameters provided by the ALS program. It reduces most of the standing army for mission planning for standard missions, and supplements an on-board AGN&C system.

Benefits of automation - An automated system would be a series of tasks for computer analysis to produce the navigation parameters for a mission. It would save schedule and labor over manual interactive processes.

Benefit of expert system - Improves analysis tools by linking them together and correlating separate databases into single database. Reduces cost.

Disadvantage - It is hard enough to delete RSC without adding an expert system which would be attacked as adding more risk.

2.2.4.7 Range Safety System [23]

The objections of safety engineers may not be easily overcome. The fact that airplanes, railroads and automobiles capable of causing significant loss of life operate today without destruct receivers with ground links may not make any difference. A Range Safety system based on ESMC 127-1 may still be the finally agreed upon approach. An expert sub-system segment of this Range Safety control approach would be utilized for assistance related to safety for operations, procedures, and design. The thought is to feed the data from the Monte Carlo Trajectory runs to drive an expert system which reviews the output data and looks for anomalies/ problems. Specific areas it might look for are: uncovered/ insufficient link signal strength, proximity to populated areas, etc., i.e. everything the range safety expert does prior to approving the launch.

Benefit of automation -- If we can reduce range support, one man-year per year.

Benefit of expert system - None.

Disadvantage - This problem has typically been one of the toughest political nuts to crack.

2.2.4.8 Command and Control Scheduler [24]

There appears to be wide spread support for a non-real time scheduling system which models the time duration and resources required for ground operations. By making this system also have expert system capabilities, historical data can be used to refine schedules, and resource management should benefit.

Benefit of automation - Better planning/ reduction of scheduling support by two heads, and increased launch crew efficiency through more micro-management of their affairs.

Benefit of expert system - The historical data can be used to improve the systems validity over time.

Disadvantage - This micro-management may not justify this system the same way earned value accounting does today.

2.2.4.9 Support for the Decision to Launch [25]

This expert system would survey (analyze) incoming real-time data from environmental monitoring stations against established criteria for launch go/no-go decision. Established criteria for launch maintained current in database. Real time environmental data covering weather, radar on surrounding air/ship intrusion into vehicle safety zones, launch window/time constraints will be directed into common expert system knowledge base.

The knowledge base will maintain a history of past failures of all flown and static tested vehicles launched from designated site. These failures shall incorporate human decision errors as well as actual vehicle failures. This expert system will aid in the launch decision process by assisting the test conductor with non-prejudiced decision information. Ideally a fully implemented system would have the ability to automatically override the go for launch decision and force a countdown hold. The decision ability of this expert system will not be susceptible to launch schedule pressures.

Benefit of automation - If this system can avoid the launch failures which seem to have been caused by deficiencies in this area, it could increase launch success by 1 to 2 per cent. This would be a very significant cost savings. Examples of this type of failure were the SS/Challenger and Atlas/Centaur-68 with its lightning strike.

Benefit of expert system - Maintenance of the software would be improved. Also heuristics data could employed as a knowledge base was developed.

Disadvantage - None.

2.2.4.10 System Wide Event Correlation [26]

Given a sub-system anomaly, this expert system correlates with all sub-system events for more rapid identification of operation anomalies.

Benefit of automation - Enhances system troubleshooting during tests and can significantly improve anomaly identification time.

Benefit of expert system - Complexities due to the number of subsystem interactions and types of sympathetic effects would be beyond conventional systems' ability to process and to discern nominal behaviors vs. abnormal behaviors.

Disadvantage - None.

2.2.4.11 Vehicle Test Conductor/Scheduler [27]

This non-real time system schedules vehicle timelines and performs procedure planning. This system will perform checklists on the vehicle and GSE status prior to test start to ensure proper configuration control on controls and automatic paper documentation processing.

Benefit of automation - This task is performed manually today requiring the use of technicians and system engineers.

Benefit of expert system - Day-to-day operations variations would make a procedural system extremely expensive and time consuming.

Disadvantage - Potential loss of flexibility.

2.2.4.12 Facilities Manager [28]

This is a non-real time system performing GSE management, consumables management, maintenance, calibration, pad and ground systems refurb scheduling, logistics management. It will integrate its activities with the vehicle test conductor / scheduler. Some subsystems will be automated for data input only.

Benefit of automation - Reduces manpower loading on ground personnel to manage GSE operations.

Benefit of expert system - Day-to-day operations variations would make a procedural system extremely expensive and time consuming.

Disadvantage - System must be flexible or cost constraints will render it ineffective.

2.2.4.13 Mission Design Automation [29]

This system will generate timelines, run simulations for flight design, validate flight data load and generate requirements. This system will be segmented functionally into several expert systems.

Benefit of automation - Significant manpower is required for recurring mission design. This will automate many analysis and paper processes.

Benefit of expert system - This system is principally a data-driven system based on mission specifics. Maintenance of this complex and changing software would be simplified.

Disadvantage - None.

2.2.4.14 Range Safety System and Recovery Operations [30]

The range safety system develops range safety requirements and pre-flight set up by using specific rules, guidelines and procedure knowledge. It performs recovery operations by integrating with recovery GSE for booster engine recovery, and with mission control for failure abort trajectory constraints.

Benefit of automation - This is typically a labor intensive task compounded by safety concerns.

Benefit of expert system - This system is principally a data driven system based on mission specifics.

Disadvantage - To operate it would require a highly reliable system.

2.2.4.15 Payload Manifesting [31]

The payload manifesting system schedules payloads for launch and determines launch vehicle/payload and payload/payload compatibility. It assesses mission capability and develops parameters to feed the mission design automation system.

Benefit of automation - This is typically a labor intensive task.

Benefit of expert system - System is data driven, lending itself to improved maintenance.

Disadvantage - None.

2.2.4.16 Countdown Operations System Monitor [32]

A real-time system will monitor all other countdown subsystems and manage their activities. This system will aid the launch director to alert him to potential problems or inconsistencies in reported data.

Benefit of automation - Highly critical task currently being done in a very distributed fashion.

Benefit of expert system - Highly data driven system. Large safety improvement.

Disadvantage - High reliability required.

2.2.4.17 Telemetry / Landlines Checks and Assignments [33]

This is a real-time system to monitor status on the health of the communication systems and make initial and on-the-fly assignments for recording systems. It would report to the launch countdown monitor.

Benefit of automation - Highly critical task currently done manually.

Benefit of expert system - Portions of the decision logic is data driven.

Disadvantage - High reliability required.

2.3 ASSESSMENT METHODOLOGY

2.3.1 A Knowledge-based System Assessment Methodology

The methodology described in this document is a general purpose technique for assessing the benefits and risks of applying knowledge-based systems (KBS) techniques to candidate onboard and ground support systems. It is not assumed that automation is suited to knowledge-based techniques (as opposed to conventional software engineering), and so this suitability is also assessed.

2.3.1.1 Methodology Overview

The assessment methodology takes input from both the domain experts and the knowledge engineers. Although the final output is an integrated evaluation of high-level benefits and risks, the intermediate results are saved for subsequent review. Thus all the details are retained for accountability, explanation and possible re-assessment. The output of the assessment is formatted in such a way that a decision maker can easily compare the evaluation of a number of candidate systems and select the best ones.

The methodology is table-driven and parts can be automated as a computer program.

2.3.1.2 Approach

The problem of candidate ranking is one of data reduction: to transform the multidimensional space of candidates with their features, impacts, costs, risks and payoffs into a linear ordering. Such a reduction necessarily involves loss of detail, even though this methodology does not actually resulting a linear ordering – that stage is left up to the final arbiters. The most important requirements in such a reduction are consistency and accountability, which are best achieved by splitting the assessment into self-contained stages which can be reviewed independently. Furthermore, by first analyzing each candidate as a unit, independently of the others, and then at the next stage analyzing its impact on the overall system, we avoid comparing apples and oranges; for example, a candidate may have a high operational risk (a unit attribute) but a low safety risk (an integrated attribute) if it is not a critical system.

2.3.1.3 Definitions of Attributes

The following unit attributes are assessed:

- *KBS Suitability* — Should the task be automated by a Knowledge-Based System, as opposed to a conventional system
- *Implementation Cost* — One-time cost of development, both manpower and hardware
- *Operational Cost*— Cost of operating the final system, both manpower and hardware (including maintenance)
- *Implementation Risk* — Risk that an operational system will not result from the development effort
- *Operational Risk* — Risk that the operational system will fail to function as expected or will not be robust

The following integrated attributes are assessed:

- *Operational Cost Improvement*— Over the lifetime of operation, the difference between non-automation and knowledge-based automation (includes both implementation and operational costs)
- *Turnaround Reduction* — Schedule savings resulting from automating an operation
- *Flight Safety Improvement*— Increase in safety (to man or machine) from automating an operation; a combination of the operational risk of the automated system, the degree of criticality of the operation, and the danger level when not automated.
- *KBS suitability and implementation risk avoidance* are also considered integrated attributes

Note further that in the following tables, + means "high" and – means "low",for example +N for operational cost means a high [degree of] operational cost whereas +N for operational cost improvement means a high [degree of] improvement in operational cost.

2.3.2 Assessment of Knowledge-Based Systems

Prior to the assessment described herein, it is assumed that an opportunity study has been conducted along the lines described in Reference 1. Thus, technical notes have been produced which identify and describe the candidate systems in a structured manner. A table is presented later (Section 3.1) which should be used to summarize this preliminary study.

Using the results of the opportunity study and further analysis by the domain experts and the knowledge engineers, the assessment produces three levels of evaluation for each candidate system, summarized in three tables which relate systems to attributes. The translation of the first table of low-level assessments into the intermediate table is automatic and generic, whereas the second translation is domain-specific (in this case Space Transportation Vehicle Mission Management Systems). The second translation results in the final table of top-level evaluations to be used to decide the best candidates.

First, numerous attributes are evaluated for each candidate by means of a series of Yes-No questions. Next, these low-level attributes are combined with a generic KBS impact-attribute matrix to produce an evaluation of each candidate in terms of the unit attributes of KBS suitability, implementation cost,operational cost, implementation risk and operational risk. Then this candidate-specific evaluation is synthesized with the customer's knowledge of how each candidate system fits into the whole operation to produce a relative evaluation of the candidates' overall impact on the high-level domain-specific attributes of operational cost improvement, turnaround time reduction and flight safety improvement. For completeness the attributes implementation risk avoidance and KBS suitability are included in this table for convenient review. The values in this table are scaled between -10 and +10.

The assessment is performed by both the domain experts and the knowledge engineers by entering values into the tables. The attributes can be weighted according to their impact on the higher level evaluation, however this may not be desirable because it is difficult to perform consistently across many comparisons and is difficult to track later.

The assessment methodology is summarized in Figure 2.1.

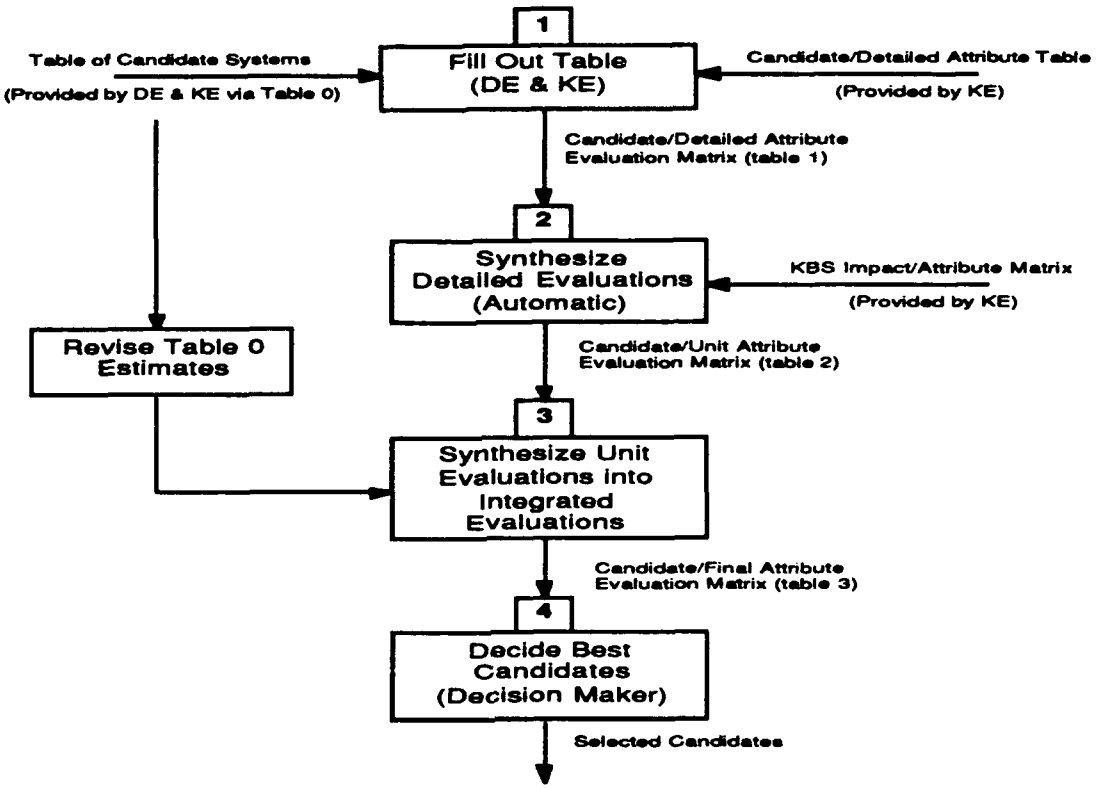


Figure 2.1—Overview of the KBS Assessment Methodology

2.3.2.1 Methodology Details

2.3.2.1.1 Prerequisites

As mentioned before, it is assumed that prior to the implementation of this assessment methodology, an opportunity study has identified a list of tasks which seem good candidates for automation by knowledge-based systems. The study will have taken into account the key drivers for cost, safety, schedule and mission success to produce a description of each candidate task which includes the following features:

- Resources required (human and others)
- Steps involved in task and relative level of complexity
- Functional, platform, performance and interface requirements
- Location in launch time-line, level of criticality for mission, danger level if not automated
- Identification of other similar tasks and similar operational systems
- Initial estimate of costs (implementation and operational) based on key drivers (resources, complexity, requirements, similarity, etc.)

This study will have been undertaken by both the domain engineers (DE) and knowledge engineers (KE). GD's Space Transportation Architecture Study report contains most of the relevant information about costs and resources. The data should be summarized in a table similar to Table 0 (Figure 2.2).

2.3.2.1.2 PROCEDURE 1 – Assess Detailed Attributes

The purpose of this stage is to fill out the detailed attribute Table 1 (Figure 2.4). The domain expert (DE) and knowledge engineer (KE) analyze the results of the opportunity study in Table 0 and answer the following questions

	Functional Area	Description	Oper. Cost/yr if not autom.	Oper. Cost/yr if automated	Implementation Cost	Mission-C[illegible] Measureme[illegible]	Danger level if not automated	Schedule savings if automated	KBS Category	KBS Example
DE/KE	DE	DE	DE	DE/KE	KE	DE	DE	DE	KE	KE
Candidate										
1	Category (e.g. ground ops) & sub-category (e.g. Flight planning, flight control, etc.)	A) References B) Requirements C) Relation to other candidates D) Degree of automation required E) Size & complexity (normalized to scale 0-10) F) Expected Lifetime	Maintenance & Manpower	Maintenance & Manpower (if not fully automated)	One time cost of S/W & H/W	Relative ratings of how critical this system is compared to other candidates (normalized to scale 0-10)	Relative rating of dangerous consequences (to man or machine) of error when not automated -- compared to ideal automation (normalized to scale 0-10)	Contribution of expected time difference to launch schedule (normalized to scale 0-10)	Based on functional description, e.g. Diagnosis Planning Monitoring Instruction Scheduling Control Prediction (maybe several)	Name of similar knowledge-based systems (with references and notes on commonality)
2										

Table 0
Figure 2.2—Summary of Opportunity Study

associated with each attribute. The questions have been designed to be mutually exclusive and complete. The nature of these questions is that they only apply to the candidate under study and require no comparison with other candidates; in other words, Table 1 should be filled out horizontally, a candidate at a time. Table 1 is of interest in its own right and represents the lowest level of assessment of each candidate.

The answeres can be binary YES (1) or NO (-1), or be scaled as certainty factors from -N to +N (typically -5 to 5) if desired. If a scaling is used, it is important to keep it consistent across all questions. The scaling of the answers could be defined as a set of selections that the domain expert and knowledge engineer use to answer the questions. The spreadsheet where the answers are stored can be used to change a text answer to a numerical value for the matrix mathematics.

BINARY SCALE		
YES = 1	NOT APPLICABLE = 0	NO = -1

SCALED CERTAINTY FACTORS		
YES = 5	NOT APPLICABLE = 0	NO = -5
MOST LIKELY = 4		NOT LIKELY = -4
PROBABLY = 3		PROBABLY NOT = -3
MAYBE = 2		MAYBE NOT = -2
I THINK SO = 1		I DON'T THINK SO = -1

Figure 2.3—Binary and Scaled Certainty Factors for the Questions

Attribute	1. Conventional Fallback	2. Non Algorithmic	3. Expert Knowledge	4. Expertise valuable	5. Unsuited to humans	6. Decomposable	7. Changing KB	8. On Critical Path	9. Existing Examples	10. State of the Art	11. Advanced UI Rqd	12. Explanation Rqd	13. Multipurpose Use	14. Many Extnl I/Fs	15. Spcl Dvlmt Rqmts	16. Can Share Mech'sms	17. Phase In Grad
DE/KE	DE	DE KE	DE KE	DE	DE	DE	DE	DE	KE	KE	DE	DE	DE	DE	KE	KE	DE
Candidate																	
1			To be filled in with a Yes (+1) or No (-1) by Domain Engineer (DE) or Knowledge Engineer (KE)														
2																	

Table 1

Figure 2.4—Detailed Attributes for Assessing Benefits and Risks of a KBS

LIST OF DETAILED ATTRIBUTE-QUESTIONS OF TABLE 1

1. Does KB automation preclude manual intervention or backup systems? This implies a high level of reliability and autonomy (not interactive).
2. Is an algorithmic (i.e., conventional) approach unsatisfactory (i.e., impossible or cost prohibitive) because the task involves combinatoric search, fuzzy rules of thumb (e.g. examples), chains of reasoning, cognitive or structural understanding, in contrast to mathematical models and simulations or operations research tools such as decision trees and optimization techniques?
3. Does the domain use expert knowledge, judgement and procedures that are well understood and available during development? That is: there is consensus about correctness; novices are routinely taught; books or manuals or test cases or experts are available to the software developers.
4. Is it desirable to compile the expertise? For example, the expertise may be expensive, scarce or likely to be unavailable in the future, or be distributed among several experts.
5. Is the task labor intensive, dangerous to do manually, or repetitive so that even experts make dumb mistakes?
6. Is the task decomposable, allowing rapid prototyping for a closed small subset and incremental development? Note that it is assumed that subsequent combination of modules will not be the major portion of the task.
7. Is the knowledge base frequently changing? Note that if the system is currently being designed then the knowledge base is being created, but if the expertise can be captured in a KBS in the design phase then it could be used for training and validation would be against simulations instead of test cases.
8. Is the project's development on the critical path for other developments?
9. Is the task similar to that of a successful existing expert system, or at least does the expected automation lend itself to known knowledge representation techniques?
10. Does the expected automation require research-level techniques, e.g. deep or hypothetical or commonsense reasoning, hybrid paradigms, learning, immature technology, deep explanations or robust advanced user interfaces?To decide this, take into account the expected technological advances in the the time-frame under consideration. Real-time systems may fall into this category, but only if the time-critical reasoning is sophisticated. Also, compare with 11 and 12.
11. Are semi-advanced AI-based user interface techniques desirable? For example, a tightly constrained subset of natural language should be understood, or non-continuous voice recognition, or some intention-based interpretation should handle incomplete or improper input.
12. Are some explanations and justifications-audit trail required, but not necessarily of a deep nature?
13. Will the system serve several purposes — e.g. simulation, design aid, training, operations?
14. Does the operational system require special (or many) external interfaces or hardware, costly host platforms, high performance (e.g.real-time), large processing units or secondary storage?
15. Does the development require special, expensive personnel or special hardware or tools?
16. Can the candidate system share modules (inference engine, knowledge base, grammar, etc.) with other candidates?
17. Can the system be phased into use gradually —i.e. will an incomplete system be considered useful even while it is being enhanced?

Figure 2.5—LIST OF DETAILED ATTRIBUTE-QUESTIONS OF TABLE 1

Another consideration that is hard to assess is political benefit: e.g. the spin-off potential, measurable payoff to skeptics, public relations, the guarantee that current practices will not be disturbed and will be welcomed by personnel.

2.3.2.1.3 PROCEDURE 2 — Synthesize Detailed Attributes Into Unit Evaluation Abacus

The low-level evaluations are synthesized into intermediate level evaluations of costs and risks by combining with a matrix which relates the impact of the low-level attributes to the intermediate level ones. The matrix entries are ±1 depending on whether there is a positive or negative impact, 0 if there is no impact. This matrix is shown in Table 2. It is expected that this matrix be constant for all assessments, however once again it may be desirable to introduce a scaling of impacts from -M to + M if appropriate.

Impact / Attribute	1. Backup	2. Non Algorithmic	3. Expert Knowledge	4. Expertise valuable	5. Unsuited to humans	6. Decomposable	7. Changing KB	8.On Critical Path	9. Existing Examples	10. State of the Art	11. Advanced UI Rqd	12. Explanation Rqd	13. Multipurpose Use	14. Many Ettnl I/Fs	15. Spcl Dvlmt Rqmts	16. Can Share Mech'sms	17. Phase In Grad
KBS Suitability	-1	+1	+1	+1	+1	+1	+1	0	+1	0	+1	+1	+1	0	0	0	+1
Implementation Cost	+1	0	0	0	0	-1	+1	0	0	+1	+1	0	0	0	+1	-1	-1
Operational Cost	0	0	0	0	-1	0	+1	0	0	0	0	0	-1	+1	0	0	0
Implementation Risk	+1	0	-1	0	0	-1	+1	+1	-1	+1	+1	0	0	0	+1	0	-1
Operational Risk	+1	0	0	0	0	0	+1	0	0	+1	+1	0	0	+1	0	0	-1

Table 2
Figure 2.6—KBS Impact - Attribute Matrix

This matrix and Table 1 are combined by aggregating the products of the YES-NO (±1) answers and the corresponding impacts (±1) for each mid-level attribute. In other words, the combination is the result of multiplying Table 1 (viewed as a matrix) by the transpose of the impact matrix in Table 2 (Figure 2.6). The result is a relative rating of candidates against the unit attributes, Table 4 (Figure 2.8).

It is easy to integrate a weighting scheme into this procedure, if it is judged that some low level attributes are more important than others. The difficulty is to construct the weights to compare several criteria, but established methods of decision analysis can be used. For example, the Analytic Heirarchy Process (AHP) constructs the weights from only pairwise comparisons of the importance of attributes (gained by interview with the knowledge engineer), and also performs a consistency check. Furthermore, it has been implemented as a commercial software product. However it is tedious to perform N*(N-1)/2 comparisons, and there is no accepted method of combining different opinions from more than one interviewee. If weights are to be utilized, each column in Table 1 will be multiplied by the corresponding weight and then this matrix used in Procedure 2. This can also be accomplished by constructing a square weight matrix as in Table 3 (where non-diagonal entries are zero and each diagonal entry is that attribute's weight, Figure 2.7) and performing a matrix multiplication. So to use the weights on the questions, the combination is the result of multiplying Table 1 (viewed .pa as a matrix) by the weight matrix (Table 3), then multiplying by the transpose of the impact matrix (Table 2). The result is a relative rating of candidates against the unit attributes, Table 4.

Attribute / Attribute	1	2	3	4	5	6	7	8	...
1	W1	0	0	0	0	0	0	0	...
2	0	W2	0	0	0	0	0	0	...
3	0	0	W3	0	0	0	0	0	...
4	0	0	0	W4	0	0	0	0	...
5	0	0	0	0	W5	0	0	0	...
6	0	0	0	0	0	W6	0	0	...
7	0	0	0	0	0	0	W7	0	...
8	0	0	0	0	0	0	0	W8	...
...	...	...	...	...	...	...	...	...	...

Table 3
Figure 2.7—Structure of Weighting Matrix for Low Level Attributes

Note that the default weighting matrix is simply the identity matrix.

	KBS Suitability	Implementation Cost	Operational Cost	Implementation Risk	Operational Risk
Scale	-12..12	-8..8	-4..4	-10..10	-6..6
Candidate					
1					
2					
3					

Table 4
Figure 2.8—Unit-Level Benefits and Risks of Applying Knowledge-Based Techniques

Note that the maximum and minimum values ("scale") for each attribute are also included so that a candidate's score can be compared to the ideal candidate. This scale is simply obtained by adding the number of non-zero items in each

row of Table 2. Of course if non-binary answers were allowed in Procedure 1, or a scaling or weighting in Table 3 was used in Procedure 2, then the max-min scale would have to be adjusted in Table 4.

To summarize, the combination algorithm is as follows:

- Generate the matrix W of weights, if required, as in Table 3.
- Perform the matrix multiplication C * W * (I transposed), where C is the completed Table 1 and I is the constant Table 2, to generate Table 4.

2.3.2.1.4 PROCEDURE 3 — Calculate Final Assessment Matrix

At this time the scores for each unit attribute have been summarized in Table 4. Now each candidate is to be rated against the other candidates as to its impact on the overall system for each of the final, integrated attributes of operational cost improvement, turnaround time reduction, flight safety improvement, implementation risk avoidance and suitability for knowledge-based system implementation. In other words, this is a vertical evaluation, an attribute at a time.

The mapping between unit attributes and final attributes takes into account many candidate-specific features derived from Table 0, which must now be revised to accommodate new understanding gained from Procedures 1 and 2. In particular, the following entries in Table 0 (which are used in Procedure 3) should be reviewed and perhaps modified:

- Size
- Oper. cost/yr if automated
- Implementation cost

Other values from Table 0 used in Procedure 3 are: the criticality of the task, its expected lifetime and its expected manual operational cost. These should not need to be revised as they depend only on the nature of the task and not on its automation.

To aid in the costs revision, the cost consistency matrix, Table 5 (Figure 2.9), compares the Table 0 dollar estimates to the Table 4 unit attributes multiplied by the size of the system (i.e. "integrated"). Each column can be scaled between 0 and 10 by a linear transformation (make the smallest value 0 and the biggest 10, X -> (X-Min)*10/(Max-Min). The scaled results and the candidate's ranks can be compared, and should approximately agree. Note, though, that costs are driven by other factors than size and the Table 1 attributes, and so a perfect agreement is not to be expected. Any surprising anomalies should be rectified by adjusting Table 0.

Candidate	Size X Implementation Cost Attribute (Abs, Scaled, Rank)	Estimated Implementation Cost (Abs, Scaled, Rank)	Size X Operational Cost Attribute (Abs, Scaled, Rank)	Estimated Operational Cost Difference/Year (Abs, Scaled, Rank)
1	From Table 0 and Table 4		From Table 0 and Table 4	
2	From Table 0 and Table 4		From Table 0 and Table 4	
3	From Table 0 and Table 4		From Table 0 and Table 4	

Table 5
Figure 2.9—Cost Consistency Matrix

Having finalized Table 0, the following mapping allows the final attributes to be calculated automatically:

KBS suitability	From Table 4
Implementation Risk Avoidance	* Value in Table 4
Operational Cost Improvement	(Manual Cost /yr – Automated Cost/yr)
	* Expected lifetime (in yrs) – Implementation cost
Turnaround Time Reduction	From Table 0
Safety Improvement	Danger level – Scaled result(-10 to +10) of Operational Risk * Criticality factor

Note that the operational cost improvement takes into account not only the expected cost of implementation and operation, but also the cost of not automating the system — even if a candidate is assessed to have a high cost factor, if it replaces a very costly alternative then it may be preferred to another candidate which is rated as low cost but may be cheap to perform manually. Similarly, the operational risk will only affect flight safety if the task the candidate performs is flight-critical.

These impact scores will be summarized in Table 6 (Figure 2.10), by scaling each column of results from -10 to 10 by making the largest (absolute) value ±10 and dividing the other values accordingly. Note that this scaling preserves the sign of the values and allows comparison with an ideal candidate (all 10's). This table will now be passed onto the final decision maker(s) for Procedure 4. It may be considered advantageous to filter out unsuitable candidates at this stage, although if an unsuitable candidate has a high payoff the conclusion might be drawn that it should be automated conventionally.

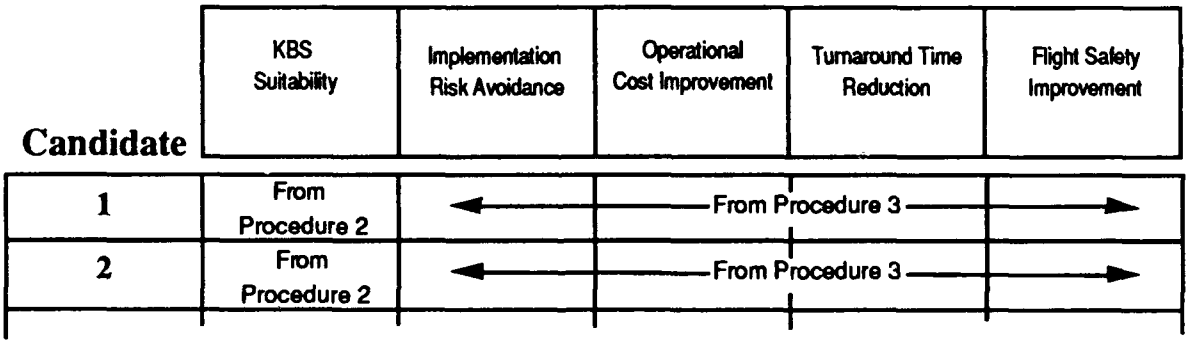


Table 6
Figure 2.10—Top Level Benefits and Risk of Applying Knowledge-Based Techniques

To summarize, Procedure 3 has the following components:

- Construct cost consistency table and revise Table 0.
- Perform the mapping from Table 0 and Table 4 into integrated attributes.
- Scale the attributes between -10 and 10 (i.e. multiply by 10/MAX) to complete Table 6.

2.3.2.1.5 PROCEDURE 4 -- Weigh Benefits and Risks of Final Assessments

At the final stage of evaluation, the decision maker(s) will be presented with the completed Table 6, which lists the candidate systems and their relative scores for suitability, benefits and risks of knowledge-based automation. The table is self-contained and so the decision maker can choose the most desirable candidate system(s) even though she has no knowledge of the nature of the candidate tasks; however the other tables and technical notes may be reviewed. It is up to the decision maker to weigh the benefits and risks in making this final choice.

Note that no attempt is made to automatically combine these final attributes into a single evaluation for each candidate, as this is not considered possible (or desirable) because of the inevitable loss of information in any such reduction. Such a decision is to be left in the hands of the final arbiter.

2.3.3 Assessment Methodology Evaluation Data and Results

- **Hardware and software implementation platform**

  This section will discuss the data that is inserted into the assessment methodology tables that were described in the previous section. We chose as the platform to run the assessment methodology, the Apple Computer Corporation's Mac II® with 5M of memory. The 5M of memory was required to allow the loading of the spreadsheet program, the assessment methodology data, and the accompanying charts for the data. We chose the Excel® spreadsheet program as the application to store the assessment methodology data in, and to perform the necessary mathematical calculations on that data. We are currently running Rev. 1.5 of the Excel program.

- **Problems with Excel and the number of files that are open**

  The Excel spreadsheet was originally generated with each of the assessment methodology tables contained in its own separate file. This caused a great deal of file manipulation and file access links whenever a table was to be opened. Or all of the files for the tables needed to be opened at the same time for the data to be plotted. So all of the main tables were copied into a single file. This eased the file access problem considerably. It also greatly reduced the number of files that needed to be open at any one time. As a side benefit from putting all of the tables in one file the length of time to perform the calculations on the data was also reduced.

- **Problems with printing the entire table on a single sheet of paper**

  When each of the assessment methodology Tables are printed on a single sheet of paper the table's text becomes so small that the table becomes extremely difficult to read. So to alleviate this problem the tables were divided into a couple of sections and then scaled to fit on a sheet of paper. Table 0 and Table 5 have been split and are printed on multiple sheets.

- **Generating charts for data clarification**

  For each of the intermediate and final selection criteria values that are generated for the candidate lists in Table 4, Table 5, and Table 6, a chart is generated to make it easy to determine the candidates with the best qualifications for that particular set of criteria.

- **Additional tables to assist in result data clarification**

  A couple of tables have also been generated to assist in sorting the candidate lists for each of the intermediate and final selection criteria values into an organized list. These extra tables have been generated for the intermediate table solutions of Table 4 and Table 5 and for the final results of Table 6. These tables list the candidates with the best qualifications of the particular selection criteria at the top of the table and the least likely candidates at the bottom.

- **The additional sort tables require a sort macro**

  The additional tables also have a associated with them a sort Macro that rearranges the list of the candidate's intermediate data and final selection values into a sorted list for easy evaluation.

- **The total number of files used to generate the assessment data**

  The total number of files that are used to generate the assessment methodology data is 21 files. Out of this there are 4 Excel spreadsheet files, 2 Excel Macro files, and 15 Excel Chart files located in 3 folders. The names of all of the chart files are listed in each of the paragraph sections about there particular table. And there file name is the same as there column header form the table from which they are produced. The file names that are used are listed in Figure 2.11.

Filename	Description
Table 0 Contains Tables 0, 1, 2, 3	4, 5, 6
Table 4.5	Contains Table 4.5
Table 5.5 - 6.5	Contains Tables 5.5, 6.5
Table 4.5 macro1	Contains Table 4.5 Sort Macro
Table 5.5 - 6.5 macro2	Contains Table 5.5, 6.5 Sort Macro
folder 4	Contains 6 Charts that are plotted form Table 4.5
folder 5	Contains 4 Charts that are plotted form Table 0
folder 6	Contains 5 Charts that are plotted form Table 0

Figure 2.11—File names used by the assessment methodology

- **Equations used in the tables**

  Each of the equations that are used throughout the Excel tables are listed after the paragraph that discuses that particular table. The names that are on the top of the table columns are used for references to its self from the other tables and as chart tittles when they are charted. The names of the columns are used to define the references because in some cases the names in the tables are different from the name that are suggested in the assessment methodology generated by Abacus. By using the column header as a reference in this manner it should make a clear understanding as to where each of the data values have come from and where each of the results are used.

2.3.3.1 Expert System Candidate Attributes and Specific Data

An additional table is generated with a numerical listing of the prospective candidates. This table is in a separate spreadsheet file but accesses the main spreadsheet file for the names of the candidates. In this way the names of the candidates need only be modified in one location and the effect of the change is seen in each of the other tables. The access of the candidate names from a single location are also done for all of the other tables as well. The key location for the candidate names are in table 0. The charts that are printed at the summation points of the assessment methodology intermediate and final conclusions, are printed with a set of numbers to describe the different candidates. This is done because: if the names of each of the candidates were printed on the charts, it would leave very little room for the plots of the data. A separate table has been added to list the associated assignment numbers printed on the charts with the names of the candidates. The order of this list of names for each of the candidates are the same for each of the charts that are printed. The only list that varies from the original list are the tables that are sorted into a list of the best qualifications to the least qualifications. The name of the expert system candidates and an explanation and/or description of each are listed in Section 2.2.

CANDIDATE EXPERT SYSTEM APPLICATIONS

01 Data Compression Analysis
02 Limit Testing
03 Critical Parameter Vehicle Surveillance (Tank Watch)
04 Hazardous Gas Identification and Safeing
05 Operator Training Simulator
06 Integrated Test Ctlr for Vehicle System C/O
07 Automatic Remote Sensor Calibration
08 Automatic Recorder Assignment
09 Launch Complex Environ. Control System
10 Operation Troubleshooting
11 Vehicle Processing Logger System
12 In-Flight Engine Perform. Monitor
13 Fluids Analysis Health Monitoring
14 Abort/Alternative Mission Modes (AGN&C)
15 Pre-Flight Test Analysis
16 Post Flight Telemetry Data Analysis
17 Flight Control Power Applic. and Monitor
18 Pneumatics, Press., and Purge Controls
19 Propellant Tanking of Vehicle
20 Engine Ignition Ground Perform. Mon.
21 Guidance Calibration
22 Mission Planning with Automated Navigation Taloring
23 Range Safety System
24 Command and Control Scheduler
25 Support for the Decision to Launch
26 System Wide Event Correlation
27 Vehicle Test Conductor/Scheduler
28 Facilities Manager
29 Mission Design Automation
30 Range Safety Sys. and Recovery Operations
31 Payload Manifesting
32 Countdown Operations System Monitor
33 Telemetry/Landlines Checks & Assignments

Figure 2.12—Candidate Expert System Applications

- **About Table 0**

  Table 0 is a set of criteria or data that has been filled out by the domain expert and knowledge base engineers. This data is used in the calculations through out the remaining tables to determine the most likely candidates for generating as an expert system. Out of the list of items for Table 0 the description of the candidates are discussed in the previous section. The KBS CATEGORY and KBS EXAMPLE of Table 0 are two columns of data that are not used in any of the equations for the other tables. So at this time they have been moved to a separate page and are listed without any corresponding data. Also in Table 0 the OPER LOC/YR AUTOMATED, and IMPLEMENTATION LOC WITHOUT ES are two columns of data that are not used in any of the calculations, but they are used as an estimation of what would be necessary to bring the system up to a point where the modification to an expert system is more likely or possible to achieve.

  The following is an explanation of the values that are generated for insertion into Table 0. Size/Complexity - The size/complexity of the expert system application candidates were generated with a scale of 1-10 referenced to a mean of 5, and a Size Reference of a medium system that would contain 100-200 rules, a typical interface to a Relational DataBase Management System (RDBMS), a single interactive graphics display, and that it would incorporate may of the software standards. - The Complexity Reference is the known procedural automation responsiveness of 1 second typical non-timing critical parsing of potential data, with quantities that typically involves tracking 500-800 parameters. Operational Costs/Yr - The operational costs were generated as the man-years of maintenance given an automated controller in hardware using traditional software and applying expert system software. Implementation Costs - The implementation costs were generated as man-years to develop and validate an expert system application using the automated hardware and application procedural software. Mission-Critical Measurement - The mission-critical measurement was generated with the following criteria, (safety and flight critical application candidates 8-10, mission critical application candidates 5-7, checkout and test vehicle application candidates 3-4, and test ground application candidates were given a 1-2 value). Expected Lifetime - The expected lifetime of each of the candidate systems were all generated to an equal value of 20 years.

- **Table 0 equations**

System Size / Complexity††	= Set by the domain expert and knowledge engineer
Oper. LOC/Yr Automated	= Set by the domain expert and knowledge engineer
Oper. LOC/Yr Expert System††	= Set by the domain expert and knowledge engineer
Implementation LOC with ES	= Set by the domain expert and knowledge engineer
Implementation LOC without ES††	= Set by the domain expert and knowledge engineer
Danger Level in not Automated†	= Set by the domain expert and knowledge engineer
Schedule Savings if Automated†	= Set by the domain expert and knowledge engineer
Expected Lifetime††	= Set by the domain expert and knowledge engineer
Mission-Critical Measurement	= Set by the domain expert and knowledge engineer
Operational Cost no Automation††	= Set by the domain expert and knowledge engineer
KBS Category†	= Set by the domain expert and knowledge engineer
KBS Example†	= Set by the domain expert and knowledge engineer

†† Have an explanation/description listed above of the values entered.

† Have no values generated at this time.

- **Table 0 values used in other tables**

Table 0	Other Tables
System Size / Complexity	Table 5
Oper. LOC/Yr Automated	Table 5
Oper. LOC/Yr Expert System	Table 5, Table 6
Implementation LOC with ES	Table 5, Table 6
Implementation LOC without ES	Table 5
Danger Level in not Automated	Table 6
Schedule Savings if Automated	Table 6
Expected Lifetime	Table 6
Mission-Critical Measurement	Table 6
Operational Cost no Automation	Table 6
KBS Category	—
KBS Example	—

Figure 2—13 Table - 0 Applies Assessment to Candidate Systems

TABLE 0 (part a)

	CANDIDATE SYSTEM	(Norm.) SYSTEM SIZE/COMPLEXITY	OPER. LOC / YR. AUTOMATED	OPER. LOC / YR. EXPERT SYSTEM	IMPLEMENTATION LOC WITH ES	IMPLEMENTATION LOC WITHOUT ES
1	Data Compression Analysis	5	100.00	40.00	1,000.00	2,500.00
2	Limit Testing	2	60.00	30.00	600.00	1,000.00
3	Critical Parameter Vehicle Surveillance	5	250.00	100.00	1,200.00	3,200.00
4	Hazardous Gas Identification and Safein	5	125.00	50.00	800.00	2,200.00
5	Operator Training Simulator	8	1,200.00	300.00	5,000.00	20,000.00
6	Integrated Test Ctlr for Vehicle System	6	1,500.00	500.00	2,800.00	8,400.00
7	Automatic Remote Sensor Calibration	3	100.00	50.00	800.00	1,800.00
8	Automatic Recorder Assignment	4	150.00	60.00	700.00	1,600.00
9	Launch Complex Environ. Control System	3	100.00	40.00	300.00	700.00
10	Operation Troubleshooting	6	1,600.00	400.00	2,400.00	7,600.00
11	Vehicle Processing Logger System	4	200.00	70.00	800.00	1,800.00
12	In-Flight Engine Perform. Monitor	6	1,280.00	420.00	2,400.00	7,200.00
13	Fluids Analysis Health Monitoring	5	1,260.00	440.00	2,400.00	6,000.00
14	Abort/Alternative Mission Modes (AGN&	6	1,600.00	500.00	5,000.00	15,000.00
15	Pre-Flight Test Analysis	8	4,000.00	1,000.00	6,000.00	24,000.00
16	Post Flight Telemetry Data Analysis	7	2,800.00	800.00	5,200.00	18,800.00
17	Flight Control Power Applic. and Monito	3	100.00	50.00	500.00	1,200.00
18	Pneumatics, Press., and Purge Controls	6	320.00	80.00	1,000.00	3,200.00
19	Propellant Tanking of Vehicle	6	820.00	240.00	2,800.00	9,400.00
20	Engine Ignition Ground Perform. Mon.	6	1,800.00	600.00	4,800.00	15,400.00
21	Guidance Calibration	5	500.00	200.00	1,200.00	3,000.00
22	Mission Planning with Automated Navig	7	3,500.00	1,000.00	5,200.00	20,800.00
23	Range Safety System	4	140.00	60.00	800.00	2,000.00
24	Command and Control Scheduler	6	2,400.00	800.00	3,600.00	12,400.00
25	Support for the Decision to Launch	6	1,500.00	500.00	4,000.00	13,000.00
26	System Wide Event Correlation	6	2,100.00	700.00	2,400.00	8,400.00
27	Vehicle Test Conductor/Scheduler	8	2,800.00	700.00	4,400.00	17,600.00
28	Facilities Manager	7	2,100.00	600.00	2,400.00	8,400.00
29	Mission Design Automation	7	1,800.00	500.00	2,800.00	11,200.00
30	Range Safety Sys. and Recovery Operatic	5	220.00	80.00	1,200.00	3,500.00
31	Payload Manifesting	5	500.00	200.00	1,200.00	3,600.00
32	Countdown Operations System Monitor	5	400.00	100.00	1,600.00	4,400.00
33	Telemetry/Landlines Checks & Assignm	5	160.00	40.00	1,200.00	3,200.00

Figure 2.13—Applied Assessment to Candidate Systems (cont.)

TABLE 0 (part b)

	CANDIDATE SYSTEM	DANGER LEVEL IF NOT AUTOMATED	SCHEDULE SAVINGS IF AUTOMATED	EXPECTED LIFETIME	MISSION-CRITICAL MEASUREMENT	OPERATIONAL COST NO AUTOMATION
1	Data Compression Analysis			20	4	100.00
2	Limit Testing			20	9	80.00
3	Critical Parameter Vehicle Surveillance (Tank Watch)			20	9	270.00
4	Hazardous Gas Identification and Safeing			20	10	165.00
5	Operator Training Simulator			20	4	1,200.00
6	Integrated Test Ctlr for Vehicle System C/O			20	5	1,500.00
7	Automatic Remote Sensor Calibration			20	3	115.00
8	Automatic Recorder Assignment			20	2	140.00
9	Launch Complex Environ. Control System			20	5	95.00
10	Operation Troubleshooting			20	1	1,250.00
11	Vehicle Processing Logger System			20	1	160.00
12	In-Flight Engine Perform. Monitor			20	10	1,260.00
13	Fluids Analysis Health Monitoring			20	9	1,100.00
14	Abort/Alternative Mission Modes (AGN&C)			20	10	1,500.00
15	Pre-Flight Test Analysis			20	4	4,000.00
16	Post Flight Telemetry Data Analysis			20	2	2,900.00
17	Flight Control Power Applic. and Monitor			20	5	120.00
18	Pneumatics, Press., and Purge Controls			20	6	260.00
19	Propellant Tanking of Vehicle			20	6	810.00
20	Engine Ignition Ground Perform. Mon.			20	7	1,925.00
21	Guidance Calibration			20	4	500.00
22	Mission Planning with Automated Navigation Taloring			20	7	4,000.00
23	Range Safety System			20	9	150.00
24	Command and Control Scheduler			20	6	2,760.00
25	Support for the Decision to Launch			20	6	1,625.00
26	System Wide Event Correlation			20	3	2,450.00
27	Vehicle Test Conductor/Scheduler			20	6	2,800.00
28	Facilities Manager			20	1	2,100.00
29	Mission Design Automation			20	1	2,000.00
30	Range Safety Sys. and Recovery Operations			20	8	240.00
31	Payload Manifesting			20	1	600.00
32	Countdown Operations System Monitor			20	6	275.00
33	Telemetry/Landlines Checks & Assignments			20	5	110.00

Figure 2.13—Applied Assessment to Candidate Systems (cont.)

2.3.3.2 Expert System Question Attributes and Weights

• About Table 1

Table 1 is an indication of how the domain expert and knowledge base engineers answered the 17 key questions on each of the projected expert system candidates. Each of the questions were to be evaluated against that particular candidate on its own merit without trying to determine its effect on the other candidates at the same time. This table can also contain a different weight for each of the candidates as long as the questions are all weighted the same for that particular candidate. This table has been generated with a set of specific selection items that could be used to generate a weighting of the questions just by answering the questions with a specific selection as well as a YES, NO, or NOT APPLICABLE answer.

If the weighted value table is to be used, the values in the table listed with Table 1 (weighted) would be modified for YES and NO from a 1 and -1 respectively to a value of 5 and -5 respectively. Otherwise the values for YES and NO would be left alone or changed back to a 1 and -1 respectively. There are actually two Table 1 listings, one with the text input weighted values and one with the converted numerical values of the text inputs. This uses a bit more memory and an extra table to implement but it allows the evaluation of a text answer rather than a numerical value to be evaluated as to wether or not the question would have any effect over the expert system candidate. Converting the text string input into of Table 1 (weighted) to a numerical value for use in the matrix multiplication of Table 4 required a little bit of planning. A table to be used for the conversion was placed under Table 1 (weighted) and because of the limited space was generated in three columns. Trying to use this table caused a equation buffer overload and forced the creation of another table that listed the names used to answer the questions in a single column. This table is listed on an individual sheet because there was no room on any other sheet to place it on where it would be relevant. If a text string is inserted into Table 1, that string is used to search the new table of selection items and then used to generate a numerical value which is put in Table 1. The addition table looks at the locations under Table 1 (weighted) for its numerical values and if the values for YES and NO are to be changed they would be changed under Table 1 (weighted) and not in the location of the new conversion table. Figure 2.14 shows a representation of what the weighted values of text could be for Table 1.

BINARY EVALUATION			
NOT APPLICABLE	= 0		
YES	= 1	NO	= -1
WEIGHTED EVALUATION			
NOT APPLICABLE	= 0		
YES	= 5	NO	= -5
MOST LIKELY	= 4	NOT LIKELY	= -4
PROBABLY	= 3	PROBABLY NOT	= -3
MAYBE	= 2	MAYBE NOT	= -2
I THINK SO	= 1	I DON'T THINK SO	= -1

Figure 2.14 Table 1 Text Input to Numerical Value Conversions

Table 1 (weighted) equations

Attribute / Question number = Set by the domain expert and knowledge engineer

Table 1 (weighted) values used in other tables

Table 1	Other Tables
Attribute / Question number	Table 1

Table 1 equations

Attribute / Question number = IF(ISERROR(MATCH(Table 1[w](text),
list of answers,0)),ERROR,
CHOOSE(MATCH(Table 1[w](text),
list of names,0),YES,MOST_LIKELY,
PROBABLY,MAYBE,THINK_SO,
DONT_THINK_SO,MAYBE_NOT,
PROBABLY_NOT,NOT_LIKELY,NO))

Table 1 values used in other tables

Table 1	Other Tables
Attribute / Question number	Table 4

TABLE 1 (weighted)

	CANDIDATE SYSTEM	ATTRIBUTE / QUESTION NUMBER																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Data Compression Analysis	YES	NO	NO	YES	NO	YES	NO	YES	YES	YES	NA	YES	NO	YES	NO	YES	NO
2	Limit Testing	YES	NO	YES	NO	NO	YES	NO	NO	YES	NO	NO	YES	NO	NO	NO	YES	NO
3	Critical Parameter Vehicle Surveillance	YES	NO	YES	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	YES	NO
4	Hazardous Gas Identification and Safein	YES	NO	YES	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	YES	NO	NO	NO
5	Operator Training Simulator	NO	YES	NO	YES	YES	YES	YES	YES	YES	NO	YES	YES	NO	NO	NO	NO	NO
6	Integrated Test Ctlr for Vehicle System	YES	YES	YES	NO	YES	YES	YES	YES	YES	NO	YES	YES	YES	NO	NO	YES	NO
7	Automatic Remote Sensor Calibration	YES	NO	YES	NO	NA	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
8	Automatic Recorder Assignment	YES	YES	YES	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
9	Launch Complex Environ. Control System	YES	NO	YES	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	YES
10	Operation Troubleshooting	NO	YES	NO	YES	NO	YES	YES	NO	YES	NO	NO	YES	NO	NO	NO	NO	YES
11	Vehicle Processing Logger System	NO	YES	YES	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	YES
12	In-Flight Engine Perform. Monitor	YES	YES	YES	NO	NA	YES	NO	YES	NO	YES	NA	YES	NO	YES	NO	YES	NO
13	Fluids Analysis Health Monitoring	YES	YES	YES	NO	NA	YES	NO	YES	NO	YES	NA	YES	YES	NO	NO	NO	NO
14	Abort/Alternative Mission Modes (AGN&(	YES	YES	NO	YES	NA	YES	YES	YES	NO	YES	NA	YES	NO	NO	NO	NO	YES
15	Pre-Flight Test Analysis	NO	YES	NO	YES	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	YES
16	Post Flight Telemetry Data Analysis	NO	YES	NO	YES	YES	YES	NO	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO
17	Flight Control Power Applic. and Monitor	YES	NO	YES	NO	NA	YES	NO	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	NO
18	Pneumatics, Press., and Purge Controls	YES	YES	YES	NO	YES	YES	NO	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	NO
19	Propellant Tanking of Vehicle	YES	YES	YES	NO	YES	YES	NO	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	NO
20	Engine Ignition Ground Perform. Mon.	YES	YES	YES	NO	YES	YES	NO	YES	NO	YES	NO	YES	NO	YES	NO	NO	NO
21	Guidance Calibration	YES	YES	YES	NO	NO	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
22	Mission Planning with Automated Navig(	NO	YES	YES	NO	YES	YES	NO	YES	YES	NO	NO	YES	YES	NO	NO	NO	YES
23	Range Safety System	YES	YES	YES	NO	NA	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
24	Command and Control Scheduler	NO	YES	YES	NO	YES	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
25	Support for the Decision to Launch	YES	YES	NO	YES	YES	YES	NO	YES	YES	NO	NA	YES	NO	NO	NO	NO	NO
26	System Wide Event Correlation	NO	YES	YES	NO	NA	YES	YES	NO	YES	NO	NA	YES	NO	NO	NO	NO	YES
27	Vehicle Test Conductor/Scheduler	YES	YES	YES	NO	YES	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	NO	YES
28	Facilities Manager	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	YES	NO	NO	NO	NO	YES
29	Mission Design Automation	NO	YES	YES	NO	YES	YES	NO	NO	YES	NO	NO	YES	NO	NO	NO	NO	YES
30	Range Safety Sys. and Recovery Operatic	YES	NO	YES	NO	NO	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
31	Payload Manifesting	NO	NO	YES	NO	YES	YES	YES	NO	YES	NO	NO	YES	NO	NO	NO	NO	YES
32	Countdown Operations System Monitor	YES	YES	YES	NO	YES	YES	NO	YES	YES	NO	YES	YES	NO	NO	NO	NO	NO
33	Telemetry/Landlines Checks & Assignm(	YES	YES	YES	NO	NO	YES	NO	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO

Unweighted responce use 1, 0, -1 ONLY	1 = YES	0 = NA	-1 = NO
For use in the case of the weighted resp use 5, 0, -5 for the YES and NO values as well as the responces found here.	4 = MOST_LIKELY 3 = PROBABLY 2 = MAYBE 1 = THINK_SO	ERR = ERROR	-4 = NOT_LIKELY -3 = PROBABLY_NOT -2 = MAYBE_NOT -1 = DONT_THINK_SO

Figure 2.15—Table -1 Weighted Attribute question effects on Candidates

TABLE 1

	CANDIDATE SYSTEM	ATTRIBUTE / QUESTION NUMBER																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Data Compression Analysis	1	-1	-1	1	-1	1	-1	1	1	1	0	1	-1	1	-1	1	-1
2	Limit Testing	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	-1	-1	1	-1
3	Critical Parameter Vehicle Surveillance	1	-1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	1	-1
4	Hazardous Gas Identification and Safein	1	-1	1	-1	1	1	-1	1	1	-1	-1	1	-1	1	-1	-1	-1
5	Operator Training Simulator	-1	1	-1	1	1	1	1	1	1	-1	1	1	-1	-1	-1	-1	-1
6	Integrated Test Ctlr for Vehicle System	1	1	1	-1	1	1	1	1	1	-1	1	1	1	-1	-1	1	-1
7	Automatic Remote Sensor Calibration	1	-1	1	-1	0	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1
8	Automatic Recorder Assignment	1	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1
9	Launch Complex Environ. Control System	1	-1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1
10	Operation Troubleshooting	-1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1	1
11	Vehicle Processing Logger System	-1	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	1
12	In-Flight Engine Perform. Monitor	1	1	1	-1	0	1	-1	1	-1	1	0	1	-1	1	-1	1	-1
13	Fluids Analysis Health Monitoring	1	1	1	-1	0	1	-1	1	-1	1	0	1	-1	1	-1	1	-1
14	Abort/Alternative Mission Modes (AGN&(	1	1	-1	1	0	1	1	1	-1	1	0	1	1	-1	-1	-1	-1
15	Pre-Flight Test Analysis	-1	1	-1	1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	1
16	Post Flight Telemetry Data Analysis	-1	1	-1	1	1	1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1
17	Flight Control Power Applic. and Monitor	1	-1	1	-1	0	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1
18	Pneumatics, Press., and Purge Controls	1	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	1	-1
19	Propellant Tanking of Vehicle	1	1	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	-1	1	-1
20	Engine Ignition Ground Perform. Mon.	1	1	1	-1	1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	-1
21	Guidance Calibration	1	1	1	-1	-1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1
22	Mission Planning with Automated Navig	-1	1	1	-1	1	1	-1	1	1	-1	-1	1	1	1	1	-1	1
23	Range Safety System	1	1	1	-1	0	1	-1	1	1	-1	-1	1	-1	1	1	-1	-1
24	Command and Control Scheduler	-1	1	1	-1	1	1	1	1	1	-1	-1	1	-1	-1	-1	-1	-1
25	Support for the Decision to Launch	1	1	-1	1	1	1	-1	1	1	-1	0	1	-1	-1	-1	1	-1
26	System Wide Event Correlation	-1	1	1	-1	0	1	1	-1	1	-1	0	1	-1	-1	-1	1	1
27	Vehicle Test Conductor/Scheduler	1	1	1	-1	1	1	1	1	1	-1	-1	1	-1	-1	-1	-1	1
28	Facilities Manager	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1
29	Mission Design Automation	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1
30	Range Safety Sys. and Recovery Operatic	1	-1	1	-1	-1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1
31	Payload Manifesting	-1	-1	1	-1	1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1	1
32	Countdown Operations System Monitor	1	1	1	-1	1	1	-1	1	1	-1	1	1	-1	-1	-1	-1	-1
33	Telemetry/Landlines Checks & Assignm	1	1	1	-1	-1	1	-1	1	1	-1	-1	1	-1	-1	-1	-1	-1

Figure 2.15—Table -1 Weighted Attribute question effects on Candidates (cont.)

2.3.3.3 Assessment Matrix Question Attributes and Weights

- **Table 2 and Table 3 Explanations And Settings**
- **About Table 2**

Table 2 is a virtually non-changeable list of effects of each of the questions on the outcome of a specific piece of data out of a required set of data. This table is used to come up with some type of a conclusion or intermediate data for a conclusion. The data in this table is defined as having a positive or negative effect for each of the questions upon a specific type of criteria. The contents of this table is defined as to whether or not the questions have any effect on the KBS SUITABILITY, IMPLEMENTATION COST, OPERATIONAL COST, IMPLEMENTATION RISK, or OPERATIONAL RISK in generating the candidate as an expert system.

Table 2 equations

Attribute / Question number	= Set by the domain expert and knowledge engineer

Table 2 values used in other tables

Table 2	Other Tables
Attribute / Question number	Table 4

TABLE 2

KEY EVALUATION CRITERIA	ATTRIBUTE / QUESTION NUMBER																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
KBS SUITABILITY	-1	1	1	1	1	1	1	0	1	0	1	1	1	0	0	0	1
IMPLEMENTATION COST	1	0	0	0	0	-1	1	0	0	1	1	0	0	0	1	-1	-1
OPERATIONAL COST	0	0	0	0	-1	0	1	0	0	0	0	0	-1	1	0	0	0
IMPLEMENTATION RISK	1	0	-1	0	0	-1	1	1	-1	1	1	0	0	0	1	0	-1
OPERATIONAL RISK	1	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	-1

Figure 2.16—Table -2 Weighted Attribute question effects on Key Criteria

- **About Table 3**

Table 3 is a set of Question weights. The question weights are used to generate a different matrix multiplication value for each of the questions. If it is determined that one or more questions should hold more weight than the another questions then the question weights should be used. This would change the effect of the questions that are listed in Table 2 and through matrix multiplication possibly the outcome of the candidate results in Table 4. Multiplying the new weight of the question in Table 2 with the candidates responses of either 1, 0, or -1 in Table 1 would change the current result values of Table 4 for the candidate and possibly its position in the computed result lists.

(Weighting of the Questions. In modifying the weights of each of the questions the following selection criteria was used. If the question was deemed to have a High, Moderate, or Low impact on the application in being generated as an expert system then the questions were weighted with a 10, 5, and 1 respectfully.)

- **Table 3 equations**

Attribute / Question number	= Set by the domain expert and knowledge engineer

- **Table 3 values used in other tables**

Table 3	Other Tables
Attribute / Question number	Table 4

TABLE 3 (part a)

	ATTRIBUTE / QUESTION WEIGHTS																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
QUESTION WEIGHTS	10	10	10	1	10	5	10	10	10	10	1	1	5	10	10	5	10

TABLE 3 (part b)

	ATTRIBUTE / QUESTION NUMBER																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10

Figure 2.17—Table -3 Weightes for Attribute questions

2.3.3.4 Synthesize Assessment Matrix Detailed Attributes

- **Table 4 Explanations And Results**
- **About Table 4**

The values in Table 4 are an intermediate conclusion table, that is derived by matrix multiplication of Table 1, Table 3, and Table 2.

[Table 4] = [Table 1] * [Table 3] * [Transpose [Table 2]]

Table 4 is further processed in Table 4.5. Table 4.5 generates the intermediate values for normalization and ranking of the candidates. This list of candidate rankings are then plotted to show the outcome of this step in the assessment methodology candidate selection. The candidate lists of Table 4.5 are further processed into a sorted list in Table 4.5 to clarify the order of the candidates. The sorted lists can then be used for easy access of the candidates that could be the best candidates to be generated as an expert system. At this step the selections would only be made from a set of intermediate values. To make a better selection the data from Table 5 and Table 6 should also be used to make a selection of a system to be generated as an expert system. The candidate lists are then plotted in their numerical order and not in their sorted order so that in all of the charts the candidates are always plotted in the same location with the same numerical value as a reference.

- **Table 4 charts are actually generated from the values in Table 4.5**

KBS SUITABILITY	
IMPLEMENTATION COST	
OPERATIONAL COST	
IMPLEMENTATION RISK	
OPERATIONAL RISK	
FINAL ASSESSMENT	= Table 4.5(Combination Actuals)

- **Table 4 equations**

KBS Suitability	= [Table 1] * [Table 3] * [Transpose[Table 2]]
Implementation Cost	= [Table 1] * [Table 3] * [Transpose[Table 2]]
Operational Cost	= [Table 1] * [Table 3] * [Transpose[Table 2]]
Implementation Risk	= [Table 1] * [Table 3] * [Transpose[Table 2]]
Operational Risk	= [Table 1] * [Table 3] * [Transpose[Table 2]]

- **Table 4 results used in other tables**

Table 5	Other Tables
KBS Suitability	Table 4.5, Table 6
Implementation Cost	Table 4.5
Operational Cost	Table 4.5
Implementation Risk	Table 4.5, Table 6
Operational Risk	Table 4.5, Table 6

TABLE 4

	CANDIDATE SYSTEM	KBS Suitability	Implementation Cost	Operational Cost	Implementation Risk	Operational Risk
1	Data Compression Analysis	-48	0	15	15	30
2	Limit Testing	-31	-21	-5	-46	-11
3	Critical Parameter Vehicle Surveillance	-11	-21	-25	-26	-11
4	Hazardous Gas Identification and Safein	-11	-11	-5	-26	9
5	Operator Training Simulator	33	-9	-5	-4	-9
6	Integrated Test Ctlr for Vehicle System	41	1	-15	-4	11
7	Automatic Remote Sensor Calibration	-21	-11	-15	-26	-11
8	Automatic Recorder Assignment	9	-11	-25	-26	-11
9	Launch Complex Environ. Control System	9	-31	-25	-46	-31
10	Operation Troubleshooting	31	-31	15	-46	-31
11	Vehicle Processing Logger System	49	-51	-25	-66	-51
12	In-Flight Engine Perform. Monitor	-20	0	5	15	30
13	Fluids Analysis Health Monitoring	-20	0	5	15	30
14	Abort/Alternative Mission Modes (AGN&	-8	30	-5	55	30
15	Pre-Flight Test Analysis	31	-51	-25	-46	-51
16	Post Flight Telemetry Data Analysis	31	-51	-25	-66	-51
17	Flight Control Power Applic. and Monitor	-21	-11	-15	-26	-11
18	Pneumatics, Press., and Purge Controls	9	-21	-25	-26	-11
19	Propellant Tanking of Vehicle	9	-21	-25	-26	-11
20	Engine Ignition Ground Perform. Mon.	-11	9	-5	14	29
21	Guidance Calibration	-11	-11	-5	-26	-11
22	Mission Planning with Automated Navig	59	-51	-35	-66	-51
23	Range Safety System	-1	-11	-15	-26	-11
24	Command and Control Scheduler	49	-11	-5	-26	-11
25	Support for the Decision to Launch	-8	-10	-25	-5	-10
26	System Wide Event Correlation	60	-30	5	-65	-30
27	Vehicle Test Conductor/Scheduler	49	-11	-5	-26	-11
28	Facilities Manager	49	-51	-25	-86	-51
29	Mission Design Automation	49	-51	-25	-86	-51
30	Range Safety Sys. and Recovery Operatic	-31	-11	-5	-26	-11
31	Payload Manifesting	49	-31	-5	-66	-31
32	Countdown Operations System Monitor	11	-9	-25	-24	-9
33	Telemetry/Landlines Checks & Assignm	-11	-11	-5	-26	-11
		Scale +- 12	Scale +- 8	Scale +- 4	Scale +- 10	Scale +- 6

Figure 2.18—Table -4 Intermediate Conclusions data matrix

- **Table 4.5 intermediate conclusion macro-equations**

KBS Suitability = (Table 4(item) - [illegible]able 4(item)))
* (10 / abs(min(Table 4(item))
- max(Table 4(item))))

(Ranking) = round((Table 4.5(item)
* (# of Candidates) / 10) + 1, 0)

Implementation Cost = (Table 4(item) - min(Table 4(item)))
* (10 / abs(min(Table 4(item))
- max(Table 4(item))))

(Ranking) = round((Table 4.5(item)
* (# of Candidates) / 10) + 1, 0)

Operational Cost = (Table 4(item) - min(Table 4(item)))
* (10 / abs(min(Table 4(item))
- max(Table 4(item))))

(Ranking) = round((Table 4.5(item)
* (# of Candidates) / 10) + 1, 0)

Implementation Risk = (Table 4(item) - min(Table 4(item)))
* (10 / abs(min(Table 4(item))
- max(Table 4(item))))

(Ranking) = round((Table 4.5(item)
* (# of Candidates) / 10) + 1, 0)

Operational Risk = (Table 4(item) - min(Table 4(item)))
* (10 / abs(min(Table 4(item))
- max(Table 4(item))))

(Ranking) = round((Table 4.5(item)
* (# of Candidates) / 10) + 1, 0)

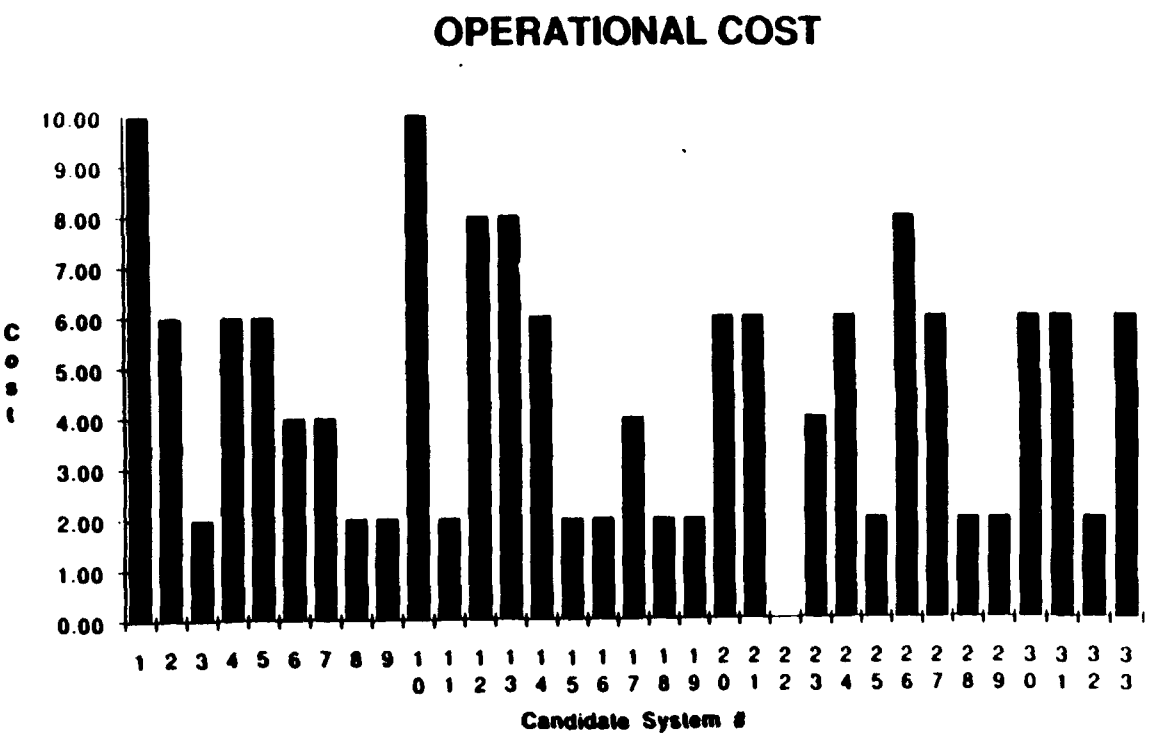
Combination Actuals = ((Table 4.5(KBS Suitability) * 1)
+ (Table 4.5(Implementation Cost) * -1)
+ (Table 4.5(Operational Cost) * -1)
+ (Table 4.5(Implementation Risk) * -1)
+ (Table 4.5(Operational Risk) * -1))

(Normalized (0-10)) = (Table 4.5(item) - min(Table 4.5(item)))
* (10 / abs(min(Table 4.5(item))
- max(Table 4.5(item))))

- **Table 4.5 sorted candidate list of intermediate conclusions and results used in other tables**

Table 4.5	Other Tables
KBS Suitability	—
(Ranking)	—
Implementation Cost	—
(Ranking)	—
Operational Cost	—
(Ranking)	—
Implementation Risk	—
(Ranking)	—
Operational Risk	—
(Ranking)	—
Combination Actuals	—
(Ranking)	—

	SORTED CANDIDATE SYSTEMS *Operational Cost*	*Sorted Actuals*
22	Mission Planning with Automated Navig	0 00
3	Critical Parameter Vehicle Surveillance	2 00
8	Automatic Recorder Assignment	2 00
9	Launch Complex Environ Control System	2 00
11	Vehicle Processing Logger System	2 00
15	Pre Flight Test Analysis	2 00
16	Post Flight Telemetry Data Analysis	2 00
18	Pneumatics, Press , and Purge Controls	2 00
19	Propellant Tanking of Vehicle	2 00
25	Support for the Decision to Launch	2 00
28	Facilities Manager	2 00
29	Mission Design Automation	2 00
32	Countdown Operations System Monitor	2 00
6	Integrated Test Ctlr for Vehicle System	4 00
7	Automatic Remote Sensor Calibration	4 00
17	Flight Control Power Applic and Monitor	4 00
23	Range Safety System	4 00
2	Limit Testing	6 00
4	Hazardous Gas Identification and Salem	6 00
5	Operator Training Simulator	6 00
14	Abort/Alternative Mission Modes (AGN&	6 00
20	Engine Ignition Ground Perform Mon	6 00
21	Guidance Calibration	6 00
24	Command and Control Scheduler	6 00
27	Vehicle Test Conductor/Scheduler	6 00
30	Range Safety Sys and Recovery Operatc	6 00
31	Payload Manifesting	6 00
33	Telemetry/Landlines Checks & Assignm	6 00
12	In-Flight Engine Perform Monitor	8 00
13	Fluids Analysis Health Monitoring	8 00
26	System Wide Event Correlation	8 00
1	Data Compression Analysis	10.00
10	Operation Troubleshooting	10.00

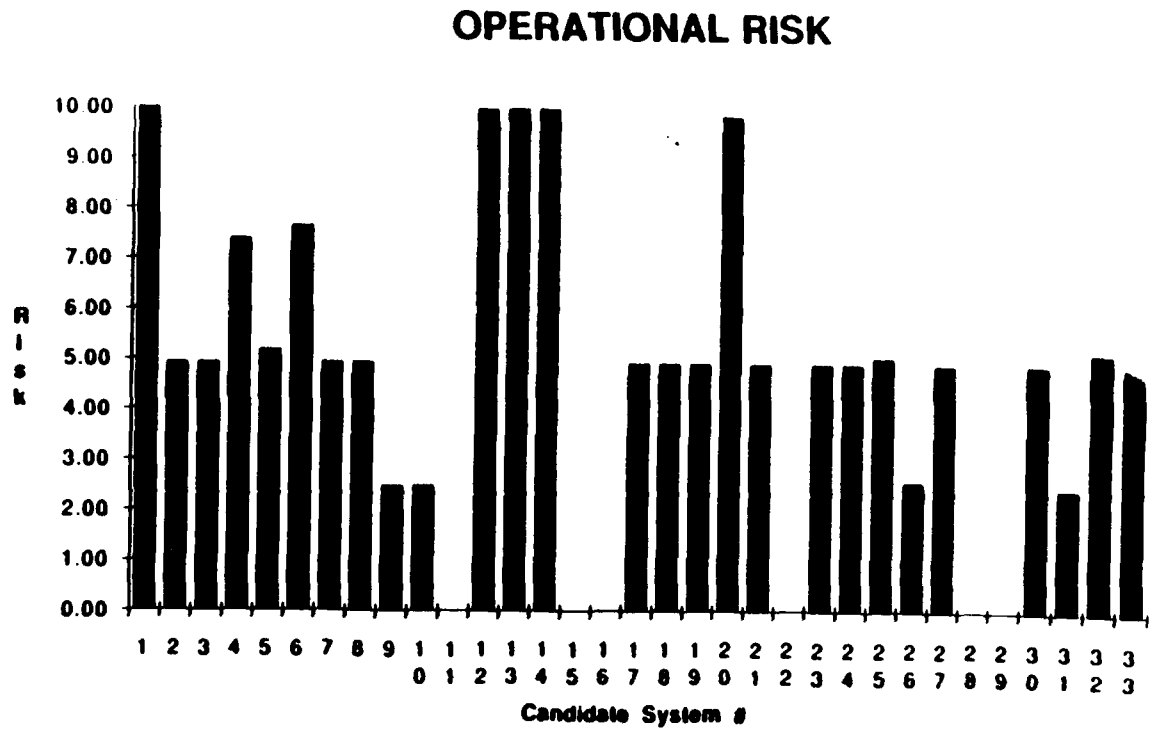


The Ops cost factor favors launch & data utilities

Note: Reverse scale

Figure 2.18—Table -4 Intermediate Conclusions data matrix (cont.)

	SORTED CANDIDATE SYSTEMS Operational Risk	Sorted Actuals
11	Vehicle Processing Logger System	0 00
15	Pre-Flight Test Analysis	0 00
16	Post Flight Telemetry Data Analysis	0 00
22	Mission Planning with Automated Navig	0 00
28	Facilities Manager	0 00
29	Mission Design Automation	0 00
9	Launch Complex Environ Control System	2 47
10	Operation Troubleshooting	2 47
31	Payload Manifesting	2 47
26	System Wide Event Correlation	2 59
2	Limit Testing	4 94
3	Critical Parameter Vehicle Surveillance	4 94
7	Automatic Remote Sensor Calibration	4 94
8	Automatic Recorder Assignment	4 94
17	Flight Control Power Applic and Monito	4 94
18	Pneumatics, Press., and Purge Controls	4 94
19	Propellant Tanking of Vehicle	4 94
21	Guidance Calibration	4 94
23	Range Safety System	4 94
24	Command and Control Scheduler	4 94
27	Vehicle Test Conductor/Scheduler	4 94
30	Range Safety Sys. and Recovery Operatic	4 94
33	Telemetry/Landlines Checks & Assignm	4 94
25	Support for the Decision to Launch	5.06
5	Operator Training Simulator	5.19
32	Countdown Operations System Monitor	5.19
4	Hazardous Gas Identification and Safetn	7 41
6	Integrated Test Ctlr for Vehicle System	7.65
20	Engine Ignition Ground Perform. Mon.	9.88
1	Data Compression Analysis	10.00
12	In-Flight Engine Perform. Monitor	10.00
13	Fluids Analysis Health Monitoring	10.00
14	Abort/Alternative Mission Modes (AGN&C	10.00

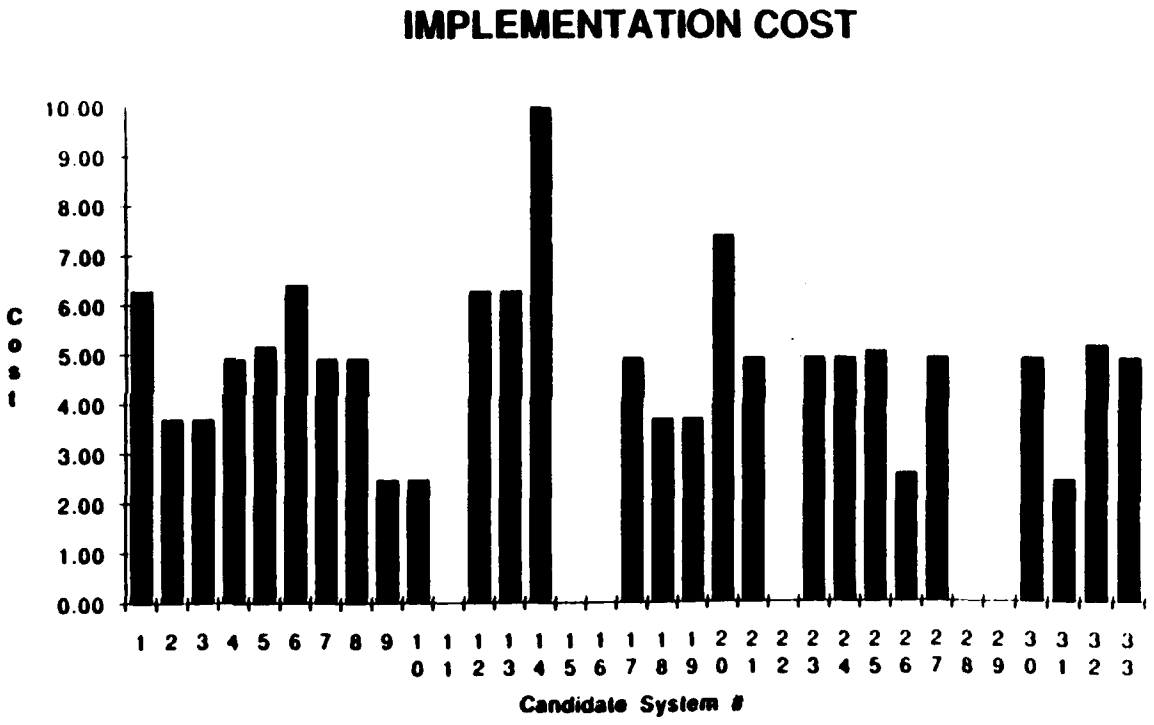


The Ops risk factor favors data analyses utilities

Note: Reverse scale

Figure 2.18—Table -4 Intermediate Conclusions data matrix (cont.)

	SORTED CANDIDATE SYSTEMS Implementation Cost	Sorted Actuals
11	Vehicle Processing Logger System	0 00
15	Pre-Flight Test Analysis	0 00
16	Post Flight Telemetry Data Analysis	0 00
22	Mission Planning with Automated Navig	0 00
28	Facilities Manager	0 00
29	Mission Design Automation	0 00
9	Launch Complex Environ Control System	2 47
10	Operation Troubleshooting	2 47
31	Payload Manifesting	2 47
26	System Wide Event Correlation	2 59
2	Limit Testing	3 70
3	Critical Parameter Vehicle Surveillance	3 70
18	Pneumatics, Press , and Purge Controls	3 70
19	Propellant Tanking of Vehicle	3 70
4	Hazardous Gas Identification and Salem	4 94
7	Automatic Remote Sensor Calibration	4 94
8	Automatic Recorder Assignment	4 94
17	Flight Control Power Applic. and Monito	4 94
21	Guidance Calibration	4 94
23	Range Safety System	4 94
24	Command and Control Scheduler	4 94
27	Vehicle Test Conductor/Scheduler	4 94
30	Range Safety Sys. and Recovery Operatio	4 94
33	Telemetry/Landlines Checks & Assignm	4 94
25	Support for the Decision to Launch	5 06
5	Operator Training Simulator	5 19
32	Countdown Operations System Monitor	5 19
1	Data Compression Analysis	6 30
12	In-Flight Engine Perform. Monitor	6 30
13	Fluids Analysis Health Monitoring	6 30
6	Integrated Test Ctlr for Vehicle System	6 42
20	Engine Ignition Ground Perform. Mon.	7 41
14	Abort/Alternative Mission Modes (AGN&	10 00

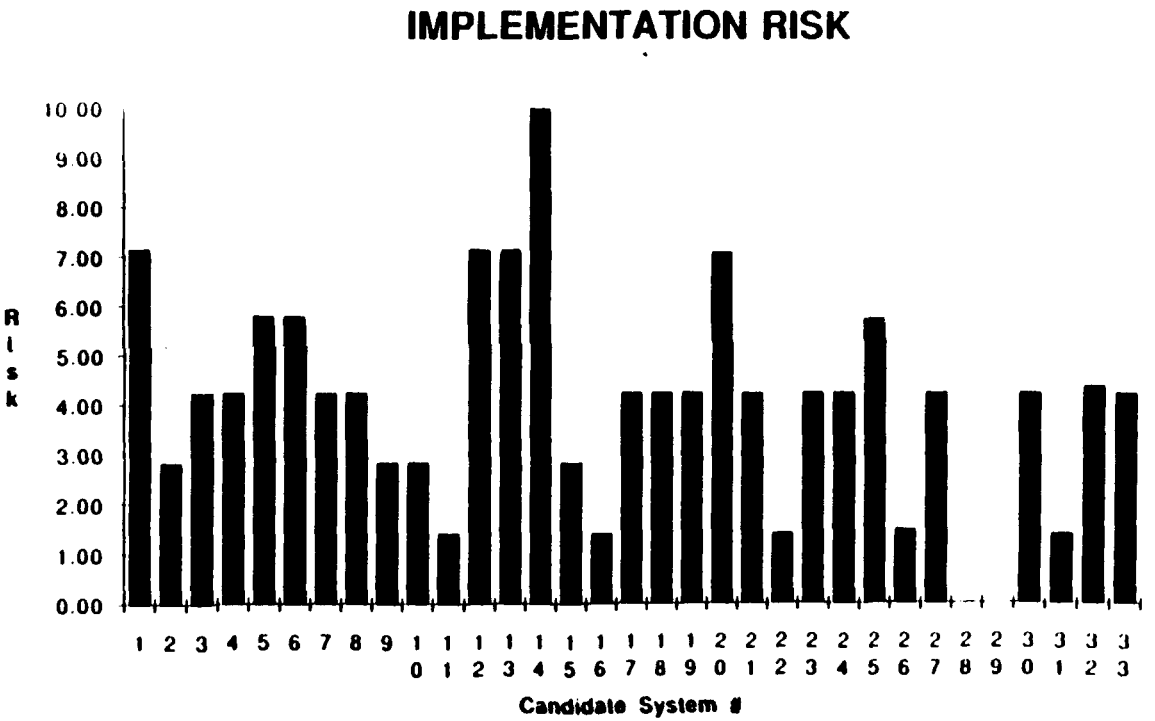


The Imp cost factor favors data processing utilities

Note: Reverse scale

Figure 2.18—Table -4 Intermediate Conclusions data matrix (cont.)

	SORTED CANDIDATE SYSTEMS Implmentation Risk	Sorted Actuals
28	Facilities Manager	0 00
29	Mission Design Automation	0 00
11	Vehicle Processing Logger System	1 42
16	Post Flight Telemetry Data Analysis	1 42
22	Mission Planning with Automated Navig	1 42
31	Payload Manifesting	1 42
26	System Wide Event Correlation	1 49
2	Limit Testing	2 84
9	Launch Complex Environ Control System	2 84
10	Operation Troubleshooting	2 84
15	Pre-Flight Test Analysis	2 84
3	Critical Parameter Vehicle Surveillance	4 26
4	Hazardous Gas Identification and Safein	4 26
7	Automatic Remote Sensor Calibration	4 26
8	Automatic Recorder Assignment	4 26
17	Flight Control Power Applic. and Monito	4 26
18	Pneumatics, Press , and Purge Controls	4 26
19	Propellant Tanking of Vehicle	4 26
21	Guidance Calibration	4 26
23	Range Safety System	4 26
24	Command and Control Scheduler	4 26
27	Vehicle Test Conductor/Scheduler	4 26
30	Range Safety Sys. and Recovery Operatic	4 26
33	Telemetry/Landlines Checks & Assignm	4 26
32	Countdown Operations System Monitor	4 40
25	Support for the Decision to Launch	5 74
5	Operator Training Simulator	5 82
6	Integrated Test Ctlr for Vehicle System	5 82
20	Engine Ignition Ground Perform Mon	7 09
1	Data Compression Analysis	7 16
12	In-Flight Engine Perform Monitor	7 16
13	Fluids Analysis Health Monitoring	7 16
14	Abort/Alternative Mission Modes (AGN&	10 00



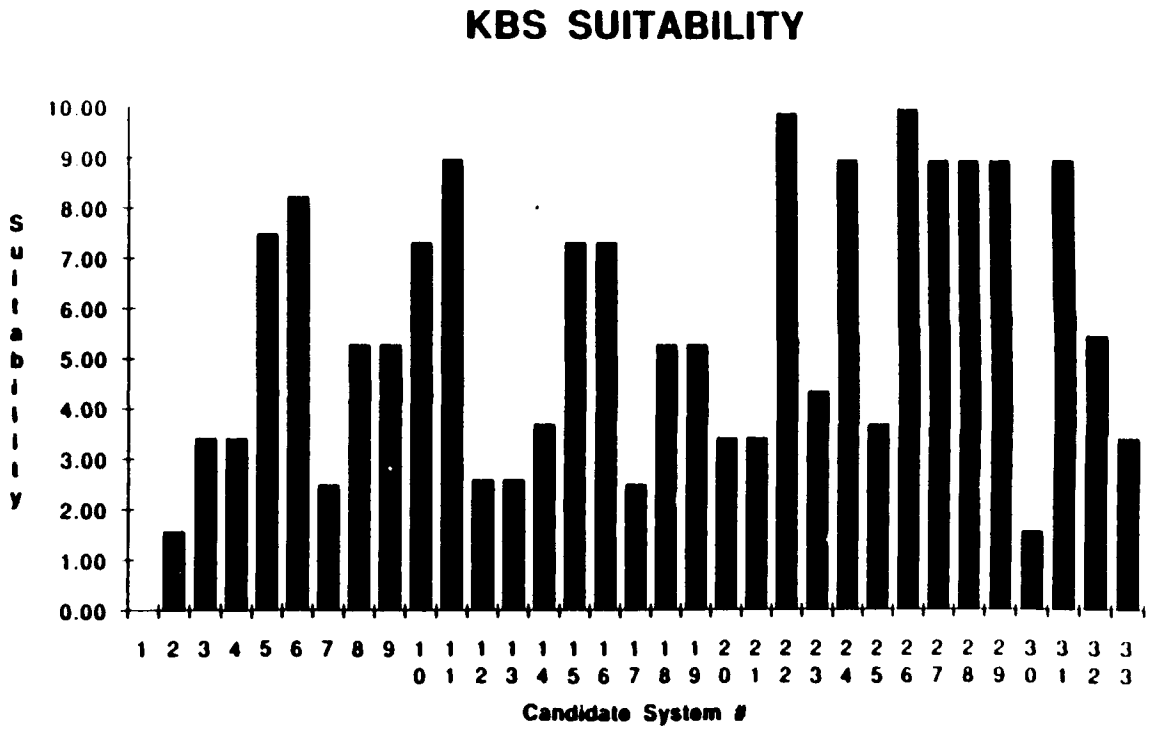
The Imp risk factor favors mission design utilities

Note: Reverse scale

Figure 2.18—Table -4 Intermediate Conclusions data matrix (cont.)

TABLE 4.5 Sorted (part a)

	SORTED CANDIDATE SYSTEMS KBS Suitability	Sorted Actuals
26	System Wide Event Correlation	10 00
22	Mission Planning with Automated Navig	9 91
11	Vehicle Processing Logger System	8 98
24	Command and Control Scheduler	8 98
27	Vehicle Test Conductor/Scheduler	8 98
28	Facilities Manager	8 98
29	Mission Design Automation	8 98
31	Payload Manifesting	8 98
6	Integrated Test Ctlr for Vehicle System	8 24
5	Operator Training Simulator	7.50
10	Operation Troubleshooting	7 31
15	Pre-Flight Test Analysis	7 31
16	Post Flight Telemetry Data Analysis	7.31
32	Countdown Operations System Monitor	5.46
8	Automatic Recorder Assignment	5.28
9	Launch Complex Environ. Control System	5.28
18	Pneumatics, Press., and Purge Controls	5.28
19	Propellant Tanking of Vehicle	5.28
23	Range Safety System	4 35
14	Abort/Alternative Mission Modes (AGN&	3.70
25	Support for the Decision to Launch	3.70
3	Critical Parameter Vehicle Surveillance	3 43
4	Hazardous Gas Identification and Salein	3 43
20	Engine Ignition Ground Perform. Mon.	3 43
21	Guidance Calibration	3 43
33	Telemetry/Landlines Checks & Assignm	3 43
12	In-Flight Engine Perform. Monitor	2 59
13	Fluids Analysis Health Monitoring	2 59
7	Automatic Remote Sensor Calibration	2 50
17	Flight Control Power Applic. and Monito	2 50
2	Limit Testing	1 57
30	Range Safety Sys. and Recovery Operatic	1 57
1	Data Compression Analysis	0 00

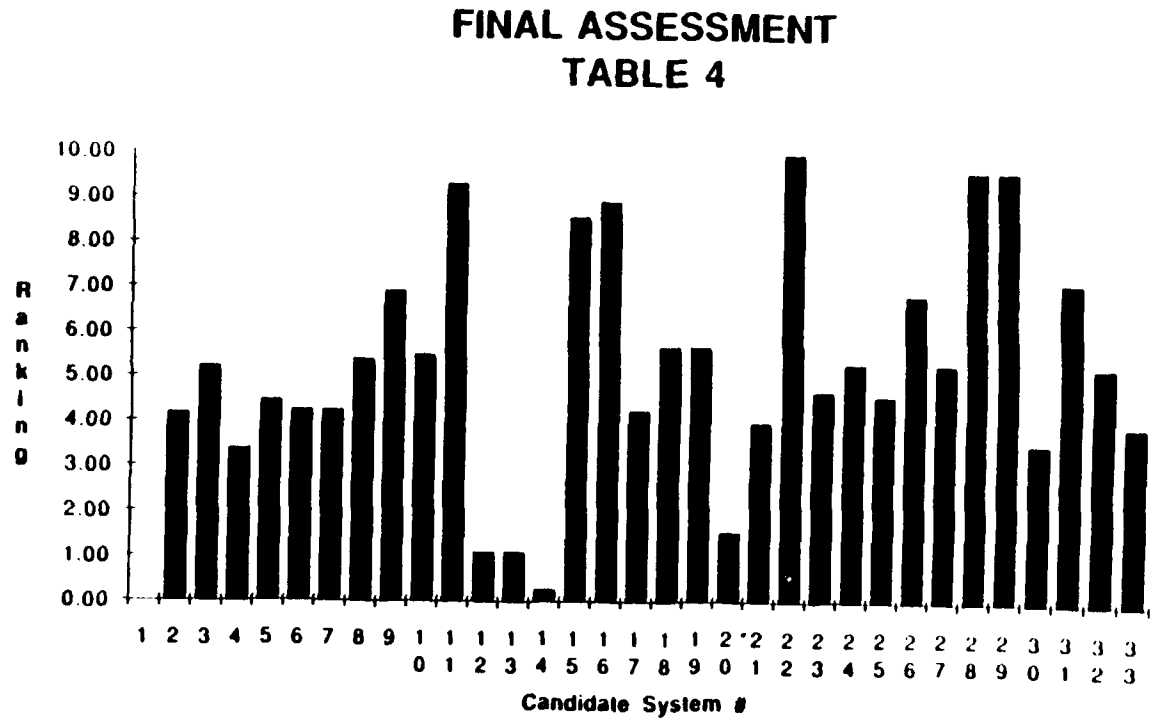


The Suitability factor favors mission operations utilities

Figure 2.18—Table -4 Intermediate Conclusions data matrix (cont.)

TABLE 4.5 Sorted (part d)

	SORTED CANDIDATE SYSTEMS Combination Ranking	Sorted Actuals
22	Mission Planning with Automated Navig	10 00
28	Facilities Manager	9 64
29	Mission Design Automation	9 64
11	Vehicle Processing Logger System	9 30
16	Post Flight Telemetry Data Analysis	8 91
15	Pre-Flight Test Analysis	8 57
31	Payload Manifesting	7 17
• 9	Launch Complex Environ Control System	6 90
26	System Wide Event Correlation	6 86
18	Pneumatics, Press, and Purge Controls	5 68
• 19	Propellant Tanking of Vehicle	5 68
10	Operation Troubleshooting	5 48
8	Automatic Recorder Assignment	5 39
24	Command and Control Scheduler	5 32
• 27	Vehicle Test Conductor/Scheduler	5 32
32	Countdown Operations System Monitor	5 28
3	Critical Parameter Vehicle Surveillance	5 24
23	Range Safety System	4 69
25	Support for the Decision to Launch	4 60
5	Operator Training Simulator	4 48
7	Automatic Remote Sensor Calibration	4 25
17	Flight Control Power Applic and Monitor	4 25
6	Integrated Test Ctlr for Vehicle System	4 25
2	Limit Testing	4 18
21	Guidance Calibration	3 99
33	Telemetry/Landlines Checks & Assignm	3 99
30	Range Safety Sys and Recovery Operatic	3 55
4	Hazardous Gas Identification and Safein	3 41
20	Engine Ignition Ground Perform Mon	1 55
12	In-Flight Engine Perform Monitor	1 09
13	Fluids Analysis Health Monitoring	1 09
14	Abort/Alternative Mission Modes (AGN&	0 28
1	Data Compression Analysis	0 00



Summation of 17 weighted methodology factors

Figure 2.18—Table -4 Intermediate Conclusions data matrix (cont.)

- **About Table 4.5 Macro**

Table 4.5 is stored in a separate file from the other tables. In order to sort the candidate list into a list in which the candidates show a ranking of top to bottom values the Excel program requires a macro to be executed. (Example: sort the implementation cost for the candidate list into a list starting with the candidate that would cost the most to implement to the candidate that would cost the least to implement)

Because of the way that the Excel program stores the data after the sort macro has been executed, the sort macro is required to be re-executed every time that the program is loaded, to ensure that the data is sorted into the correct locations for observations or for printing. The sorting macro sorts only the data and not the equations so that when the data is stored back into the file upon closing the file, the data is actually stored into the file back in the table's unsorted state. The program also takes so long to run a recalculation of all the tables every time that a value is changed in any table that the macro changes the Excel program into a manual calculation mode. This means that a manual calculation and/or the sort Macro, has to be executed after any data has been changed or after the program has been started. Otherwise, data in the tables would be inconsistent or incorrect.

Note: The Excel program operator needs to ensure that the correct table or program is active when the macro is executed or else the macro will walk through a pertinent file sorting the values into an undesirable state]

```
Record1
=CALCULATION(3,FALSE)
=SELECT("R4C25")
=CALCULATE.NOW()
=SELECT("R4C25:R36C27")
=SORT(1,"R4C27",2)
=VSCROLL(0%)
=HLINE(4)
=SELECT("R4C29:R36C31")
=SORT(1,"R4C31",2)
=HLINE(4)
=VSCROLL(0%)
=SELECT("R4C33:R36C35")
=SORT(1,"R4C35",1)
=HLINE(4)
=VSCROLL(0%)
=SELECT("R4C37:R36C39")
=SORT(1,"R4C39",1)
=HLINE(4)
=VSCROLL(0%)
=SELECT("R4C41:R36C43")
=SORT(1,"R4C43",1)
=HLINE(4)
=VSCROLL(0%)
=SELECT("R4C45:R36C47")
=SORT(1,"R4C47",1)
=HLINE(4)
=HSCROLL(3.9063%)
=HLINE(4)
=VSCROLL(0%)
=SELECT("R4C25")
=HLINE(2)
=RETURN()
```

Figure 2-19—Sort Macro to produce Table - 4.5

TABLE 4.5 (part a)

	CANDIDATE SYSTEM	KBS Suitability	Ranking
1	Data Compression Analysis	0.00	1.00
2	Limit Testing	1.57	6.00
3	Critical Parameter Vehicle Surveillance	3.43	12.00
4	Hazardous Gas Identification and Safein	3.43	12.00
5	Operator Training Simulator	7.50	25.00
6	Integrated Test Ctlr for Vehicle System	8.24	27.00
7	Automatic Remote Sensor Calibration	2.50	9.00
8	Automatic Recorder Assignment	5.28	18.00
9	Launch Complex Environ. Control System	5.28	18.00
10	Operation Troubleshooting	7.31	24.00
11	Vehicle Processing Logger System	8.98	30.00
12	In-Flight Engine Perform. Monitor	2.59	9.00
13	Fluids Analysis Health Monitoring	2.59	9.00
14	Abort/Alternative Mission Modes (AGN&	3.70	13.00
15	Pre-Flight Test Analysis	7.31	24.00
16	Post Flight Telemetry Data Analysis	7.31	24.00
17	Flight Control Power Applic. and Monito	2.50	9.00
18	Pneumatics, Press., and Purge Controls	5.28	18.00
19	Propellant Tanking of Vehicle	5.28	18.00
20	Engine Ignition Ground Perform. Mon.	3.43	12.00
21	Guidance Calibration	3.43	12.00
22	Mission Planning with Automated Navig	9.91	33.00
23	Range Safety System	4.35	15.00
24	Command and Control Scheduler	8.98	30.00
25	Support for the Decision to Launch	3.70	13.00
26	System Wide Event Correlation	10.00	33.00
27	Vehicle Test Conductor/Scheduler	8.98	30.00
28	Facilities Manager	8.98	30.00
29	Mission Design Automation	8.98	30.00
30	Range Safety Sys. and Recovery Operatic	1.57	6.00
31	Payload Manifesting	8.98	30.00
32	Countdown Operations System Monitor	5.46	18.00
33	Telemetry/Landlines Checks & Assignm	3.43	12.00

Figure 2.20—Table-4.5 Intermediate Macro Sorted Data

TABLE 4.5 (part b)

	CANDIDATE SYSTEM	Implementation Cost	Ranking	Operational Cost	Ranking
1	Data Compression Analysis	6.30	21.00	10.00	33.00
2	Limit Testing	3.70	13.00	6.00	20.00
3	Critical Parameter Vehicle Surveillance	3.70	13.00	2.00	7.00
4	Hazardous Gas Identification and Safein	4.94	17.00	6.00	20.00
5	Operator Training Simulator	5.19	18.00	6.00	20.00
6	Integrated Test Ctlr for Vehicle System	6.42	22.00	4.00	14.00
7	Automatic Remote Sensor Calibration	4.94	17.00	4.00	14.00
8	Automatic Recorder Assignment	4.94	17.00	2.00	7.00
9	Launch Complex Environ. Control System	2.47	9.00	2.00	7.00
10	Operation Troubleshooting	2.47	9.00	10.00	33.00
11	Vehicle Processing Logger System	0.00	1.00	2.00	7.00
12	In-Flight Engine Perform. Monitor	6.30	21.00	8.00	27.00
13	Fluids Analysis Health Monitoring	6.30	21.00	8.00	27.00
14	Abort/Alternative Mission Modes (AGN&S	10.00	33.00	6.00	20.00
15	Pre-Flight Test Analysis	0.00	1.00	2.00	7.00
16	Post Flight Telemetry Data Analysis	0.00	1.00	2.00	7.00
17	Flight Control Power Applic. and Monitor	4.94	17.00	4.00	14.00
18	Pneumatics, Press., and Purge Controls	3.70	13.00	2.00	7.00
19	Propellant Tanking of Vehicle	3.70	13.00	2.00	7.00
20	Engine Ignition Ground Perform. Mon.	7.41	25.00	6.00	20.00
21	Guidance Calibration	4.94	17.00	6.00	20.00
22	Mission Planning with Automated Navig	0.00	1.00	0.00	1.00
23	Range Safety System	4.94	17.00	4.00	14.00
24	Command and Control Scheduler	4.94	17.00	6.00	20.00
25	Support for the Decision to Launch	5.06	17.00	2.00	7.00
26	System Wide Event Correlation	2.59	9.00	8.00	27.00
27	Vehicle Test Conductor/Scheduler	4.94	17.00	6.00	20.00
28	Facilities Manager	0.00	1.00	2.00	7.00
29	Mission Design Automation	0.00	1.00	2.00	7.00
30	Range Safety Sys. and Recovery Operatic	4.94	17.00	6.00	20.00
31	Payload Manifesting	2.47	9.00	6.00	20.00
32	Countdown Operations System Monitor	5.19	18.00	2.00	7.00
33	Telemetry/Landlines Checks & Assignm	4.94	17.00	6.00	20.00

Figure 2.20—Table-4.5 Intermediate Macro Sorted Data (Cont.)

TABLE 4.5 (part c)

	CANDIDATE SYSTEM	Implementation Risk	Ranking	Operational Risk	Ranking
1	Data Compression Analysis	7.16	24.00	10.00	33.00
2	Limit Testing	2.84	10.00	4.94	17.00
3	Critical Parameter Vehicle Surveillance	4.26	15.00	4.94	17.00
4	Hazardous Gas Identification and Safein	4.26	15.00	7.41	25.00
5	Operator Training Simulator	5.82	20.00	5.19	18.00
6	Integrated Test Ctlr for Vehicle System	5.82	20.00	7.65	25.00
7	Automatic Remote Sensor Calibration	4.26	15.00	4.94	17.00
8	Automatic Recorder Assignment	4.26	15.00	4.94	17.00
9	Launch Complex Environ. Control System	2.84	10.00	2.47	9.00
10	Operation Troubleshooting	2.84	10.00	2.47	9.00
11	Vehicle Processing Logger System	1.42	6.00	0.00	1.00
12	In-Flight Engine Perform. Monitor	7.16	24.00	10.00	33.00
13	Fluids Analysis Health Monitoring	7.16	24.00	10.00	33.00
14	Abort/Alternative Mission Modes (AGN&	10.00	33.00	10.00	33.00
15	Pre-Flight Test Analysis	2.84	10.00	0.00	1.00
16	Post Flight Telemetry Data Analysis	1.42	6.00	0.00	1.00
17	Flight Control Power Applic. and Monito	4.26	15.00	4.94	17.00
18	Pneumatics, Press., and Purge Controls	4.26	15.00	4.94	17.00
19	Propellant Tanking of Vehicle	4.26	15.00	4.94	17.00
20	Engine Ignition Ground Perform. Mon.	7.09	24.00	9.88	33.00
21	Guidance Calibration	4.26	15.00	4.94	17.00
22	Mission Planning with Automated Navig	1.42	6.00	0.00	1.00
23	Range Safety System	4.26	15.00	4.94	17.00
24	Command and Control Scheduler	4.26	15.00	4.94	17.00
25	Support for the Decision to Launch	5.74	19.00	5.06	17.00
26	System Wide Event Correlation	1.49	6.00	2.59	9.00
27	Vehicle Test Conductor/Scheduler	4.26	15.00	4.94	17.00
28	Facilities Manager	0.00	1.00	0.00	1.00
29	Mission Design Automation	0.00	1.00	0.00	1.00
30	Range Safety Sys. and Recovery Operatic	4.26	15.00	4.94	17.00
31	Payload Manifesting	1.42	6.00	2.47	9.00
32	Countdown Operations System Monitor	4.40	15.00	5.19	18.00
33	Telemetry/Landlines Checks & Assignm	4.26	15.00	4.94	17.00

Figure 2.20—Table-4.5 Intermediate Macro Sorted Data (Cont.)

TABLE 4.5 (part d)

	CANDIDATE SYSTEM	Combination (0-10)	Combination Actual
1	Data Compression Analysis	0.00	-6.69
2	Limit Testing	4.18	-3.18
3	Critical Parameter Vehicle Surveillance	5.24	-2.29
4	Hazardous Gas Identification and Safein	3.41	-3.84
5	Operator Training Simulator	4.48	-2.94
6	Integrated Test Ctlr for Vehicle System	4.25	-3.13
7	Automatic Remote Sensor Calibration	4.25	-3.13
8	Automatic Recorder Assignment	5.39	-2.17
9	Launch Complex Environ. Control System	6.90	-0.90
10	Operation Troubleshooting	5.48	-2.09
11	Vehicle Processing Logger System	9.30	1.11
12	In-Flight Engine Perform. Monitor	1.09	-5.77
13	Fluids Analysis Health Monitoring	1.09	-5.77
14	Abort/Alternative Mission Modes (AGN&	0.28	-6.46
15	Pre-Flight Test Analysis	8.57	0.50
16	Post Flight Telemetry Data Analysis	8.91	0.78
17	Flight Control Power Applic. and Monito	4.25	-3.13
18	Pneumatics, Press., and Purge Controls	5.68	-1.92
19	Propellant Tanking of Vehicle	5.68	-1.92
20	Engine Ignition Ground Perform. Mon.	1.55	-5.39
21	Guidance Calibration	3.99	-3.34
22	Mission Planning with Automated Navig	10.00	1.70
23	Range Safety System	4.69	-2.76
24	Command and Control Scheduler	5.32	-2.23
25	Support for the Decision to Launch	4.60	-2.83
26	System Wide Event Correlation	6.86	-0.93
27	Vehicle Test Conductor/Scheduler	5.32	-2.23
28	Facilities Manager	9.64	1.40
29	Mission Design Automation	9.64	1.40
30	Range Safety Sys. and Recovery Operatic	3.55	-3.71
31	Payload Manifesting	7.17	-0.68
32	Countdown Operations System Monitor	5.28	-2.26
33	Telemetry/Landlines Checks & Assignm	3.99	-3.34

TABLE 4.5 Sorted (part a)

	SORTED CANDIDATE SYSTEMS KBS Suitability	Sorted Actuals
26	System Wide Event Correlation	10.00
22	Mission Planning with Automated Navig	9.91
11	Vehicle Processing Logger System	8.98
24	Command and Control Scheduler	8.98
27	Vehicle Test Conductor/Scheduler	8.98
28	Facilities Manager	8.98
29	Mission Design Automation	8.98
31	Payload Manifesting	8.98
6	Integrated Test Ctlr for Vehicle System	8.24
5	Operator Training Simulator	7.50
10	Operation Troubleshooting	7.31
15	Pre-Flight Test Analysis	7.31
16	Post Flight Telemetry Data Analysis	7.31
32	Countdown Operations System Monitor	5.46
8	Automatic Recorder Assignment	5.28
9	Launch Complex Environ. Control System	5.28
18	Pneumatics, Press., and Purge Controls	5.28
19	Propellant Tanking of Vehicle	5.28
23	Range Safety System	4.35
14	Abort/Alternative Mission Modes (AGN&	3.70
25	Support for the Decision to Launch	3.70
3	Critical Parameter Vehicle Surveillance	3.43
4	Hazardous Gas Identification and Safein	3.43
20	Engine Ignition Ground Perform. Mon.	3.43
21	Guidance Calibration	3.43
33	Telemetry/Landlines Checks & Assignm	3.43
12	In-Flight Engine Perform. Monitor	2.59
13	Fluids Analysis Health Monitoring	2.59
7	Automatic Remote Sensor Calibration	2.50
17	Flight Control Power Applic. and Monito	2.50
2	Limit Testing	1.57
30	Range Safety Sys. and Recovery Operatic	1.57
1	Data Compression Analysis	0.00

Figure 2.20—Table-4.5 Intermediate Macro Sorted Data (Cont.)

TABLE 4.5 Sorted (part b)

	SORTED CANDIDATE SYSTEMS Implementation Cost	Sorted Actuals
11	Vehicle Processing Logger System	0.00
15	Pre-Flight Test Analysis	0.00
16	Post Flight Telemetry Data Analysis	0.00
22	Mission Planning with Automated Navig	0.00
28	Facilities Manager	0.00
29	Mission Design Automation	0.00
9	Launch Complex Environ. Control System	2.47
10	Operation Troubleshooting	2.47
31	Payload Manifesting	2.47
26	System Wide Event Correlation	2.59
2	Limit Testing	3.70
3	Critical Parameter Vehicle Surveillance	3.70
18	Pneumatics, Press., and Purge Controls	3.70
19	Propellant Tanking of Vehicle	3.70
4	Hazardous Gas Identification and Safein	4.94
7	Automatic Remote Sensor Calibration	4.94
8	Automatic Recorder Assignment	4.94
17	Flight Control Power Applic. and Monitor	4.94
21	Guidance Calibration	4.94
23	Range Safety System	4.94
24	Command and Control Scheduler	4.94
27	Vehicle Test Conductor/Scheduler	4.94
30	Range Safety Sys. and Recovery Operatic	4.94
33	Telemetry/Landlines Checks & Assignm	4.94
25	Support for the Decision to Launch	5.06
5	Operator Training Simulator	5.19
32	Countdown Operations System Monitor	5.19
1	Data Compression Analysis	6.30
12	In-Flight Engine Perform. Monitor	6.30
13	Fluids Analysis Health Monitoring	6.30
6	Integrated Test Ctlr for Vehicle System	6.42
20	Engine Ignition Ground Perform. Mon.	7.41
14	Abort/Alternative Mission Modes (AGN&	10.00

	SORTED CANDIDATE SYSTEMS Operational Cost	Sorted Actuals
22	Mission Planning with Automated Navig	0.00
3	Critical Parameter Vehicle Surveillance	2.00
8	Automatic Recorder Assignment	2.00
9	Launch Complex Environ. Control System	2.00
11	Vehicle Processing Logger System	2.00
15	Pre-Flight Test Analysis	2.00
16	Post Flight Telemetry Data Analysis	2.00
18	Pneumatics, Press., and Purge Controls	2.00
19	Propellant Tanking of Vehicle	2.00
25	Support for the Decision to Launch	2.00
28	Facilities Manager	2.00
29	Mission Design Automation	2.00
32	Countdown Operations System Monitor	2.00
6	Integrated Test Ctlr for Vehicle System	4.00
7	Automatic Remote Sensor Calibration	4.00
17	Flight Control Power Applic. and Monitor	4.00
23	Range Safety System	4.00
2	Limit Testing	6.00
4	Hazardous Gas Identification and Safein	6.00
5	Operator Training Simulator	6.00
14	Abort/Alternative Mission Modes (AGN&	6.00
20	Engine Ignition Ground Perform. Mon.	6.00
21	Guidance Calibration	6.00
24	Command and Control Scheduler	6.00
27	Vehicle Test Conductor/Scheduler	6.00
30	Range Safety Sys. and Recovery Operatic	6.00
31	Payload Manifesting	6.00
33	Telemetry/Landlines Checks & Assignm	6.00
12	In-Flight Engine Perform. Monitor	8.00
13	Fluids Analysis Health Monitoring	8.00
26	System Wide Event Correlation	8.00
1	Data Compression Analysis	10.00
10	Operation Troubleshooting	10.00

Figure 2.20—Table-4.5 Intermediate Macro Sorted Data (Cont.)

TABLE 4.5 Sorted (part c)

	SORTED CANDIDATE SYSTEMS Implementation Risk	Sorted Actuals
28	Facilities Manager	0.00
29	Mission Design Automation	0.00
11	Vehicle Processing Logger System	1.42
16	Post Flight Telemetry Data Analysis	1.42
22	Mission Planning with Automated Navig	1.42
31	Payload Manifesting	1.42
26	System Wide Event Correlation	1.49
2	Limit Testing	2.84
9	Launch Complex Environ. Control System	2.84
10	Operation Troubleshooting	2.84
15	Pre-Flight Test Analysis	2.84
3	Critical Parameter Vehicle Surveillance	4.26
4	Hazardous Gas Identification and Safein	4.26
7	Automatic Remote Sensor Calibration	4.26
8	Automatic Recorder Assignment	4.26
17	Flight Control Power Applic. and Monitor	4.26
18	Pneumatics, Press., and Purge Controls	4.26
19	Propellant Tanking of Vehicle	4.26
21	Guidance Calibration	4.26
23	Range Safety System	4.26
24	Command and Control Scheduler	4.26
27	Vehicle Test Conductor/Scheduler	4.26
30	Range Safety Sys. and Recovery Operatic	4.26
33	Telemetry/Landlines Checks & Assignm	4.26
32	Countdown Operations System Monitor	4.40
25	Support for the Decision to Launch	5.74
5	Operator Training Simulator	5.82
6	Integrated Test Ctlr for Vehicle System	5.82
20	Engine Ignition Ground Perform. Mon.	7.09
1	Data Compression Analysis	7.16
12	In-Flight Engine Perform. Monitor	7.16
13	Fluids Analysis Health Monitoring	7.16
14	Abort/Alternative Mission Modes (AGN&	10.00

	SORTED CANDIDATE SYSTEMS Operational Risk	Sorted Actuals
11	Vehicle Processing Logger System	0.00
15	Pre-Flight Test Analysis	0.00
16	Post Flight Telemetry Data Analysis	0.00
22	Mission Planning with Automated Navig	0.00
28	Facilities Manager	0.00
29	Mission Design Automation	0.00
9	Launch Complex Environ. Control System	2.47
10	Operation Troubleshooting	2.47
31	Payload Manifesting	2.47
26	System Wide Event Correlation	2.59
2	Limit Testing	4.94
3	Critical Parameter Vehicle Surveillance	4.94
7	Automatic Remote Sensor Calibration	4.94
8	Automatic Recorder Assignment	4.94
17	Flight Control Power Applic. and Monitor	4.94
18	Pneumatics, Press., and Purge Controls	4.94
19	Propellant Tanking of Vehicle	4.94
21	Guidance Calibration	4.94
23	Range Safety System	4.94
24	Command and Control Scheduler	4.94
27	Vehicle Test Conductor/Scheduler	4.94
30	Range Safety Sys. and Recovery Operatic	4.94
33	Telemetry/Landlines Checks & Assignm	4.94
25	Support for the Decision to Launch	5.06
5	Operator Training Simulator	5.19
32	Countdown Operations System Monitor	5.19
4	Hazardous Gas Identification and Safein	7.41
6	Integrated Test Ctlr for Vehicle System	7.65
20	Engine Ignition Ground Perform. Mon.	9.88
1	Data Compression Analysis	10.00
12	In-Flight Engine Perform. Monitor	10.00
13	Fluids Analysis Health Monitoring	10.00
14	Abort/Alternative Mission Modes (AGN&	10.00

Figure 2.20—Table 4.5 Intermediate Macro Sorted Data (Cont.)

TABLE 4.5 Sorted (part d)

	SORTED CANDIDATE SYSTEMS Combination Ranking	*Sorted Actuals*
22	Mission Planning with Automated Navig	10.00
28	Facilities Manager	9.64
29	Mission Design Automation	9.64
11	Vehicle Processing Logger System	9.30
16	Post Flight Telemetry Data Analysis	8.91
15	Pre-Flight Test Analysis	8.57
31	Payload Manifesting	7.17
9	Launch Complex Environ. Control System	6.90
26	System Wide Event Correlation	6.86
18	Pneumatics, Press., and Purge Controls	5.68
19	Propellant Tanking of Vehicle	5.68
10	Operation Troubleshooting	5.48
8	Automatic Recorder Assignment	5.39
24	Command and Control Scheduler	5.32
27	Vehicle Test Conductor/Scheduler	5.32
32	Countdown Operations System Monitor	5.28
3	Critical Parameter Vehicle Surveillance	5.24
23	Range Safety System	4.69
25	Support for the Decision to Launch	4.60
5	Operator Training Simulator	4.48
7	Automatic Remote Sensor Calibration	4.25
17	Flight Control Power Applic. and Monitor	4.25
6	Integrated Test Ctlr for Vehicle System	4.25
2	Limit Testing	4.18
21	Guidance Calibration	3.99
33	Telemetry/Landlines Checks & Assignm	3.99
30	Range Safety Sys. and Recovery Operatic	3.55
4	Hazardous Gas Identification and Safein	3.41
20	Engine Ignition Ground Perform. Mon.	1.55
12	In-Flight Engine Perform. Monitor	1.09
13	Fluids Analysis Health Monitoring	1.09
14	Abort/Alternative Mission Modes (AGN&	0.28
1	Data Compression Analysis	0.00

Figure 2.20—Table-4.5 Intermediate Macro Sorted Data (Cont.)

2.3.3.5 Calculate Assessment Matrix Cost Benefits & Risks

- **Table 5 and Table 6 Explanations And Results**
- **About Table 5**

Table 5 cost data conclusions. The data in Table 5 is derived from the cost and size data in Table 0. The data results in Table 5 are a direct function of the values that are inserted into Table 0. The questions and their weights of Table 2 and Table 4 have no effect over the data in Table 5. The candidate's tabulations of Table 5 are then plotted as they were in Table 4. A list of the charts for Table 5 are listed below. The conclusions of Table 5 are further processed in Table 5.5 into a sorted list of the candidates showing the candidates with the highest cost as the largest values. This table is of little use except to see at a glance the candidate's location in that particular list or column of the table.

- **Table 5 charts**

  SIZE * IMPLEMENTATION COST

  SIZE * OPERATIONAL COST

  COMBINED IMPLEMENTATION COST + OPERATIONAL COST

  SAVINGS (Expert System vs ALGORITHMIC)

- **Table 5 results used in other tables**

Table 5	Other Tables
Size * Implementation Cost	
(Absolute)	—
(Normalized (0-10))	—
(Ranking)	—
Implementation Cost	
(Absolute)	—
(Normalized (0-10))	—
(Ranking)	—
Size * Operational Cost	
(Absolute)	—
(Normalized (0-10))	—
(Ranking)	—
1st Yr Estimated savings (ES vs. Algorithmic)	
(Absolute)	—
(Normalized (0-10))	—
(Ranking)	—
Size * Costs Combined	—

- **Table 5 equations**

Size * Implementation Cost

(Absolute) = Table 0(System Size / Complexity)
* Table 0(Implementation LOC with ES)

(Normalized (0-10)) = (Table 5(item) - min(Table 5(item)))
* (10 / abs(min(Table 5(item))
- max(Table 5(item))))

(Ranking) = round((Table 5(item)
* (# of Candidates) / 10) + 1, 0)

Implementation Cost

(Absolute) = Table 0(Implementation LOC with ES)

(Normalized (0-10)) = (Table 5(item) - min(Table 5(item)))
* (10 / abs(min(Table 5(item))
- max(Table 5(item))))

(Ranking) = round((Table 5(item)
* (# of Candidates) / 10) + 1, 0)

Size * Operational Cost

(Absolute) = Table 0(System Size / Complexity)
* Table 0(Oper. LOC/Yr with ES)

(Normalized (0-10)) = (Table 5(item) - min(Table 5(item)))
* (10 / abs(min(Table 5(item))
- max(Table 5(item))))

(Ranking) = round((Table 5(item)
* (# of Candidates) / 10) + 1, 0)

1st Yr Estimated savings (ES vs. Algorithmic)

(Absolute) = (Table 0(Oper. LOC/Yr Automated)
- Table 0(Oper. LOC/Yr with ES))
+ (Table 0(Implementation LOC without ES)
- Table 0(Implementation LOC with ES))

(Normalized (0-10)) = (Table 5(item) - min(Table 5(item)))
* (10 / abs(min(Table 5(item))
- max(Table 5(item))))

(Ranking) = round((Table 5(item)
* (# of Candidates) / 10) + 1, 0)

Size * Costs Combined = (Table 5(Size * Implementation Cost)
+ Table 5(Size * Operational Cost)) / 2

TABLE 5 (part a)

	CANDIDATE SYSTEM	SIZE * IMPLEMENTATION COST			IMPLEMENTATION COST		
		Absolute	Normalized(0-10)	Ranking	Absolute	Normalized(0-10)	Ranking
1	Data Compression Analysis	5,000.00	0.87	4.00	1,000.00	1.23	5.00
2	Limit Testing	1,200.00	0.06	1.00	600.00	0.53	3.00
3	Critical Parameter Vehicle Surveillance	6,000.00	1.08	4.00	1,200.00	1.58	6.00
4	Hazardous Gas Identification and Safein	4,000.00	0.66	3.00	800.00	0.88	4.00
5	Operator Training Simulator	40,000.00	8.30	28.00	5,000.00	8.25	27.00
6	Integrated Test Ctlr for Vehicle System	16,800.00	3.38	12.00	2,800.00	4.39	15.00
7	Automatic Remote Sensor Calibration	2,400.00	0.32	2.00	800.00	0.88	4.00
8	Automatic Recorder Assignment	2,800.00	0.40	2.00	700.00	0.70	3.00
9	Launch Complex Environ. Control System	900.00	0.00	1.00	300.00	0.00	1.00
10	Operation Troubleshooting	14,400.00	2.87	10.00	2,400.00	3.68	13.00
11	Vehicle Processing Logger System	3,200.00	0.49	3.00	800.00	0.88	4.00
12	In-Flight Engine Perform. Monitor	14,400.00	2.87	10.00	2,400.00	3.68	13.00
13	Fluids Analysis Health Monitoring	12,000.00	2.36	9.00	2,400.00	3.68	13.00
14	Abort/Alternative Mission Modes (AGN&(	30,000.00	6.18	21.00	5,000.00	8.25	27.00
15	Pre-Flight Test Analysis	48,000.00	10.00	33.00	6,000.00	10.00	33.00
16	Post Flight Telemetry Data Analysis	36,400.00	7.54	25.00	5,200.00	8.60	29.00
17	Flight Control Power Applic. and Monitor	1,500.00	0.13	1.00	500.00	0.35	2.00
18	Pneumatics, Press., and Purge Controls	6,000.00	1.08	4.00	1,000.00	1.23	5.00
19	Propellant Tanking of Vehicle	16,800.00	3.38	12.00	2,800.00	4.39	15.00
20	Engine Ignition Ground Perform. Mon.	28,800.00	5.92	20.00	4,800.00	7.89	26.00
21	Guidance Calibration	6,000.00	1.08	4.00	1,200.00	1.58	6.00
22	Mission Planning with Automated Navig	36,400.00	7.54	25.00	5,200.00	8.60	29.00
23	Range Safety System	3,200.00	0.49	3.00	800.00	0.88	4.00
24	Command and Control Scheduler	21,600.00	4.39	15.00	3,600.00	5.79	20.00
25	Support for the Decision to Launch	24,000.00	4.90	17.00	4,000.00	6.49	22.00
26	System Wide Event Correlation	14,400.00	2.87	10.00	2,400.00	3.68	13.00
27	Vehicle Test Conductor/Scheduler	35,200.00	7.28	24.00	4,400.00	7.19	24.00
28	Facilities Manager	16,800.00	3.38	12.00	2,400.00	3.68	13.00
29	Mission Design Automation	19,600.00	3.97	14.00	2,800.00	4.39	15.00
30	Range Safety Sys. and Recovery Operatic	6,000.00	1.08	4.00	1,200.00	1.58	6.00
31	Payload Manifesting	6,000.00	1.08	4.00	1,200.00	1.58	6.00
32	Countdown Operations System Monitor	8,000.00	1.51	6.00	1,600.00	2.28	8.00
33	Telemetry/Landlines Checks & Assignm	6,000.00	1.08	4.00	1,200.00	1.58	6.00

Figure 2.21—Table-5 Cost data intermediate results

TABLE 5 (part b)

	CANDIDATE SYSTEM	SIZE * OPERATIONAL COST			1st Yr Estimated Savings (ES vs. Algorithmic)		
		Absolute	Normalized(0-10)	Ranking	Absolute	Normalized(0-10)	Ranking
1	Data Compression Analysis	200.00	0.18	2.00	1,560.00	0.55	3.00
2	Limit Testing	60.00	0.00	1.00	430.00	0.00	1.00
3	Critical Parameter Vehicle Surveillance	500.00	0.55	3.00	2,150.00	0.84	4.00
4	Hazardous Gas Identification and Safein	250.00	0.24	2.00	1,475.00	0.51	3.00
5	Operator Training Simulator	2,400.00	2.95	10.00	15,900.00	7.52	25.00
6	Integrated Test Ctlr for Vehicle System	3,000.00	3.70	13.00	6,600.00	3.00	11.00
7	Automatic Remote Sensor Calibration	150.00	0.11	1.00	1,050.00	0.30	2.00
8	Automatic Recorder Assignment	240.00	0.23	2.00	990.00	0.27	2.00
9	Launch Complex Environ. Control System	120.00	0.08	1.00	460.00	0.01	1.00
10	Operation Troubleshooting	2,400.00	2.95	10.00	6,400.00	2.90	10.00
11	Vehicle Processing Logger System	280.00	0.28	2.00	1,130.00	0.34	2.00
12	In-Flight Engine Perform. Monitor	2,520.00	3.10	11.00	5,660.00	2.54	9.00
13	Fluids Analysis Health Monitoring	2,200.00	2.70	10.00	4,420.00	1.94	7.00
14	Abort/Alternative Mission Modes (AGN&	3,000.00	3.70	13.00	11,100.00	5.19	18.00
15	Pre-Flight Test Analysis	8,000.00	10.00	33.00	21,000.00	10.00	33.00
16	Post Flight Telemetry Data Analysis	5,600.00	6.98	23.00	15,600.00	7.37	25.00
17	Flight Control Power Applic. and Monitor	150.00	0.11	1.00	750.00	0.16	1.00
18	Pneumatics, Press., and Purge Controls	480.00	0.53	3.00	2,440.00	0.98	4.00
19	Propellant Tanking of Vehicle	1,440.00	1.74	7.00	7,180.00	3.28	12.00
20	Engine Ignition Ground Perform. Mon.	3,600.00	4.46	15.00	11,800.00	5.53	19.00
21	Guidance Calibration	1,000.00	1.18	5.00	2,100.00	0.81	4.00
22	Mission Planning with Automated Navig	7,000.00	8.74	29.00	18,100.00	8.59	28.00
23	Range Safety System	240.00	0.23	2.00	1,280.00	0.41	2.00
24	Command and Control Scheduler	4,800.00	5.97	20.00	10,400.00	4.85	17.00
25	Support for the Decision to Launch	3,000.00	3.70	13.00	10,000.00	4.65	16.00
26	System Wide Event Correlation	4,200.00	5.21	18.00	7,400.00	3.39	12.00
27	Vehicle Test Conductor/Scheduler	5,600.00	6.98	23.00	15,300.00	7.23	24.00
28	Facilities Manager	4,200.00	5.21	18.00	7,500.00	3.44	12.00
29	Mission Design Automation	3,500.00	4.33	15.00	9,700.00	4.51	15.00
30	Range Safety Sys. and Recovery Operatic	400.00	0.43	2.00	2,440.00	0.98	4.00
31	Payload Manifesting	1,000.00	1.18	5.00	2,700.00	1.10	5.00
32	Countdown Operations System Monitor	500.00	0.55	3.00	3,100.00	1.30	5.00
33	Telemetry/Landlines Checks & Assignm	200.00	0.18	2.00	2,120.00	0.82	4.00

Figure 2.21—Table-5 Cost data intermediate results (Cont.)

TABLE 5 (part c)

	CANDIDATE SYSTEM	size * costs Combined
1	Data Compression Analysis	0.52
2	Limit Testing	0.03
3	Critical Parameter Vehicle Surveillance	0.82
4	Hazardous Gas Identification and Safein	0.45
5	Operator Training Simulator	5.62
6	Integrated Test Ctlr for Vehicle System	3.54
7	Automatic Remote Sensor Calibration	0.22
8	Automatic Recorder Assignment	0.32
9	Launch Complex Environ. Control System	0.04
10	Operation Troubleshooting	2.91
11	Vehicle Processing Logger System	0.38
12	In-Flight Engine Perform. Monitor	2.98
13	Fluids Analysis Health Monitoring	2.53
14	Abort/Alternative Mission Modes (AGN&	4.94
15	Pre-Flight Test Analysis	10.00
16	Post Flight Telemetry Data Analysis	7.26
17	Flight Control Power Applic. and Monitor	0.12
18	Pneumatics, Press., and Purge Controls	0.81
19	Propellant Tanking of Vehicle	2.56
20	Engine Ignition Ground Perform. Mon.	5.19
21	Guidance Calibration	1.13
22	Mission Planning with Automated Navig	8.14
23	Range Safety System	0.36
24	Command and Control Scheduler	5.18
25	Support for the Decision to Launch	4.30
26	System Wide Event Correlation	4.04
27	Vehicle Test Conductor/Scheduler	7.13
28	Facilities Manager	4.29
29	Mission Design Automation	4.15
30	Range Safety Sys. and Recovery Operatic	0.76
31	Payload Manifesting	1.13
32	Countdown Operations System Monitor	1.03
33	Telemetry/Landlines Checks & Assignm	0.63

Figure 2.21—Table-5 Cost data intermediate results (Cont.)

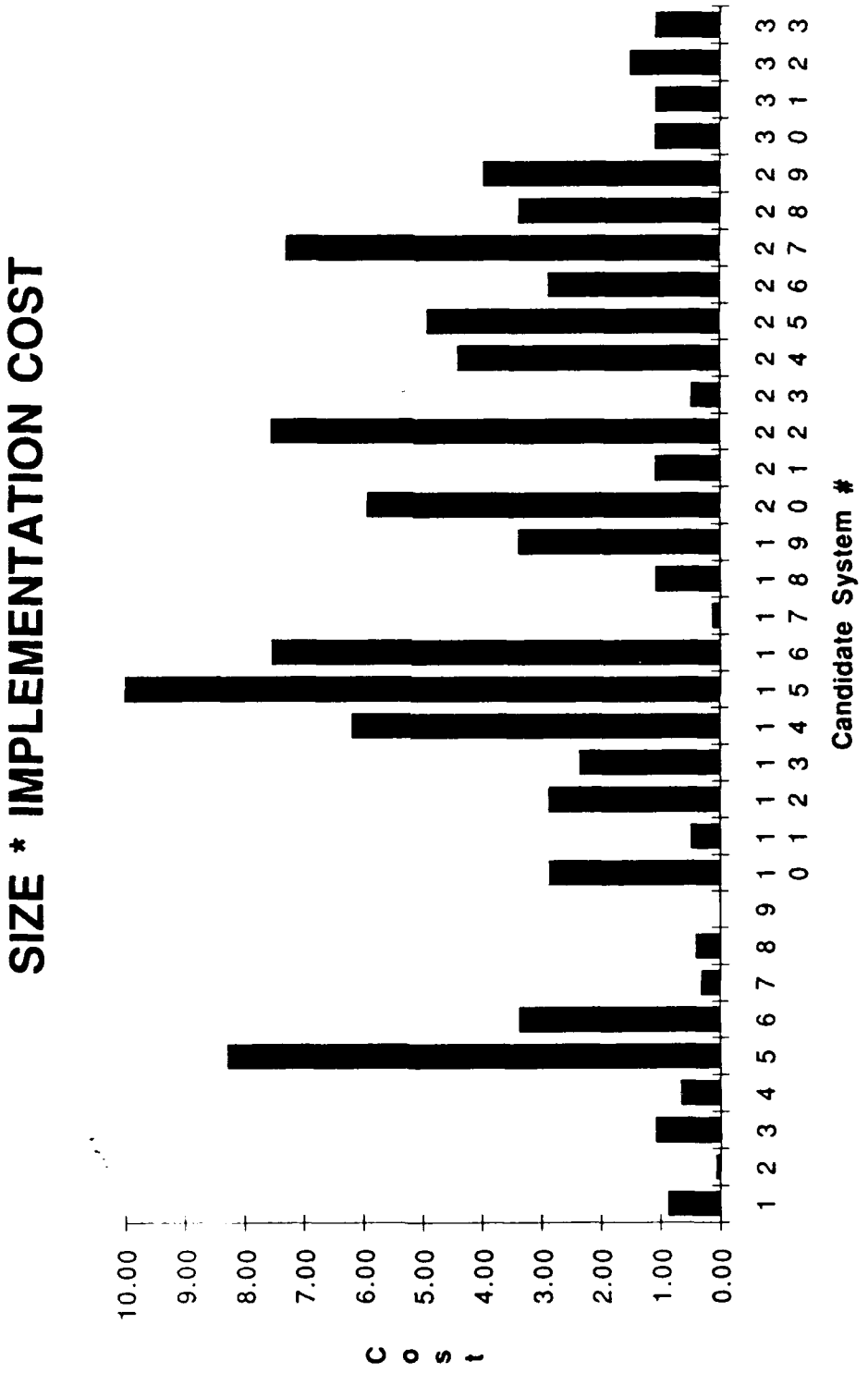
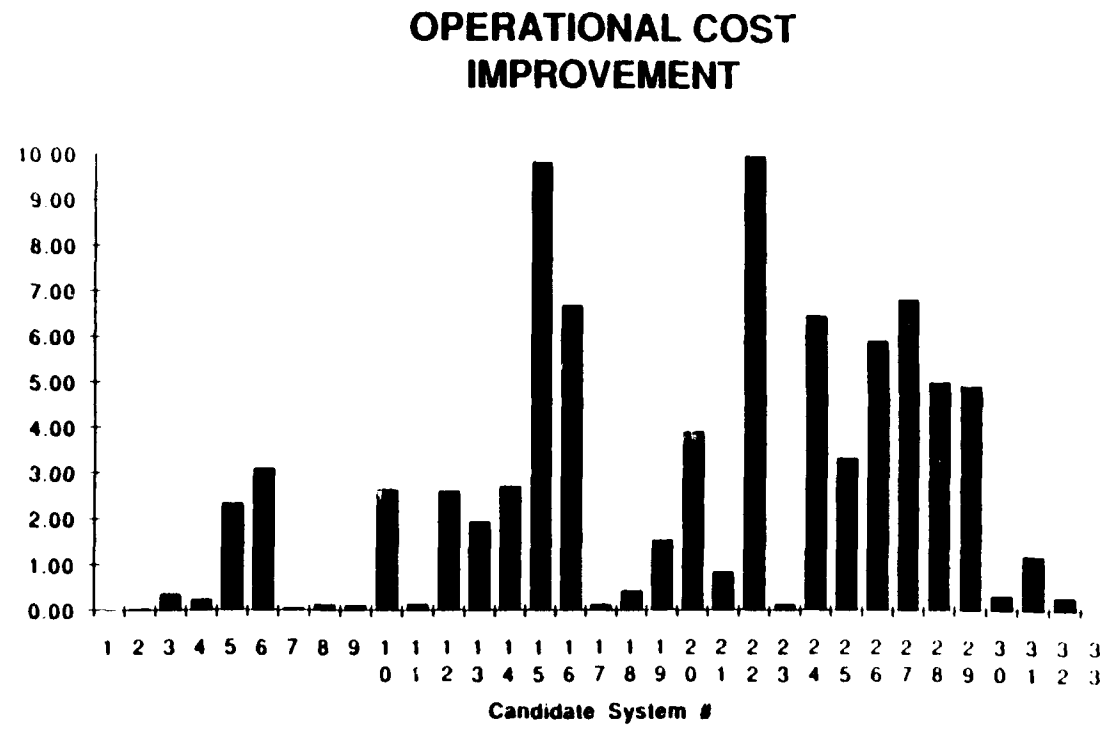


Figure 2.21—Table-5 Cost data intermediate results (Cont.)

	CANDIDATE SYSTEM	OPER COST IMPROVEMENT
22	Mission Planning with Automated Navig	10 00
15	Pre Flight Test Analysis	9 85
27	Vehicle Test Conduction/Scheduler	6 85
16	Post Flight Telemetry Data Analysis	6 70
24	Command and Control Scheduler	6 48
26	System Wide Event Correlation	5 93
28	Facilities Manager	5 02
29	Mission Design Automation	4 95
20	Engine Ignition Ground Perform Mon	3 94
25	Support for the Decision to Launch	3 35
6	Integrated Test Cllr for Vehicle System	3 11
14	Abort/Alternative Mission Modes (AGN&C	2 71
10	Operation Troubleshooting	2 64
12	In-Flight Engine Perform Monitor	2 60
5	Operator Training Simulator	2 34
13	Fluids Analysis Health Monitoring	1 94
19	Propellant Tanking of Vehicle	1 54
31	Payload Manifesting	1 21
21	Guidance Calibration	0 84
18	Pneumatics, Press, and Purge Controls	0 44
3	Critical Parameter Vehicle Surveillance	0 37
30	Range Safety Sys and Recovery Operatic	0 33
32	Countdown Operations System Monitor	0 31
4	Hazardous Gas Identification and Safetn	0 24
11	Vehicle Processing Logger System	0 15
23	Range Safety System	0 15
8	Automatic Recorder Assignment	0 13
17	Flight Control Power Applic and Monito	0 13
9	Launch Complex Environ Control System	0 11
7	Automatic Remote Sensor Calibration	0 05
2	Limit Testing	0 04
1	Data Compression Analysis	0 00
33	Telemetry/Landlines Checks & Assignme	0 00



The Cost improve. favors launch processing utilities

Figure 2.21—Table-5 Cost data intermediate results (Cont.)

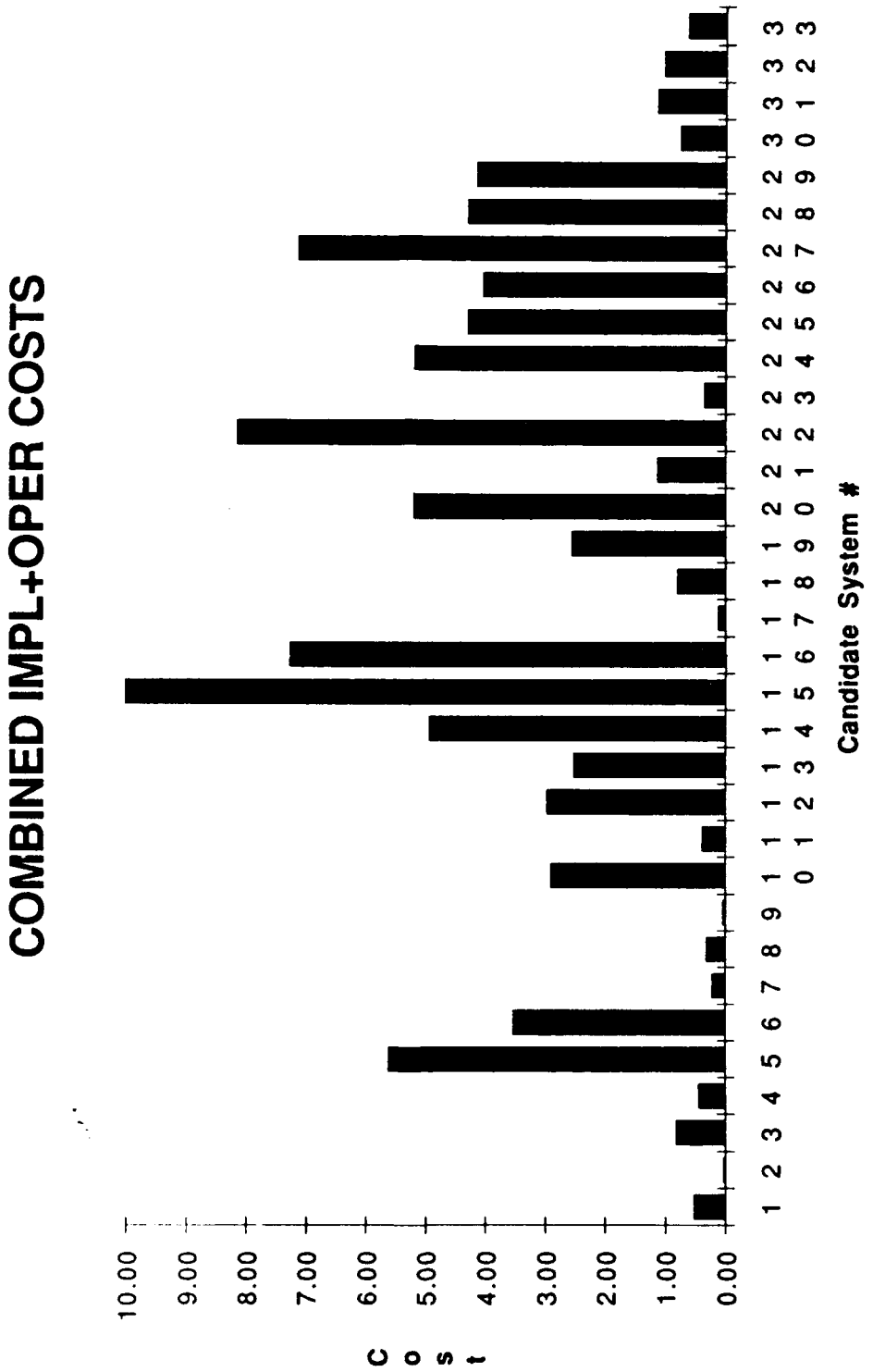


Figure 2.21—Table-5 Cost data intermediate results (Cont.)

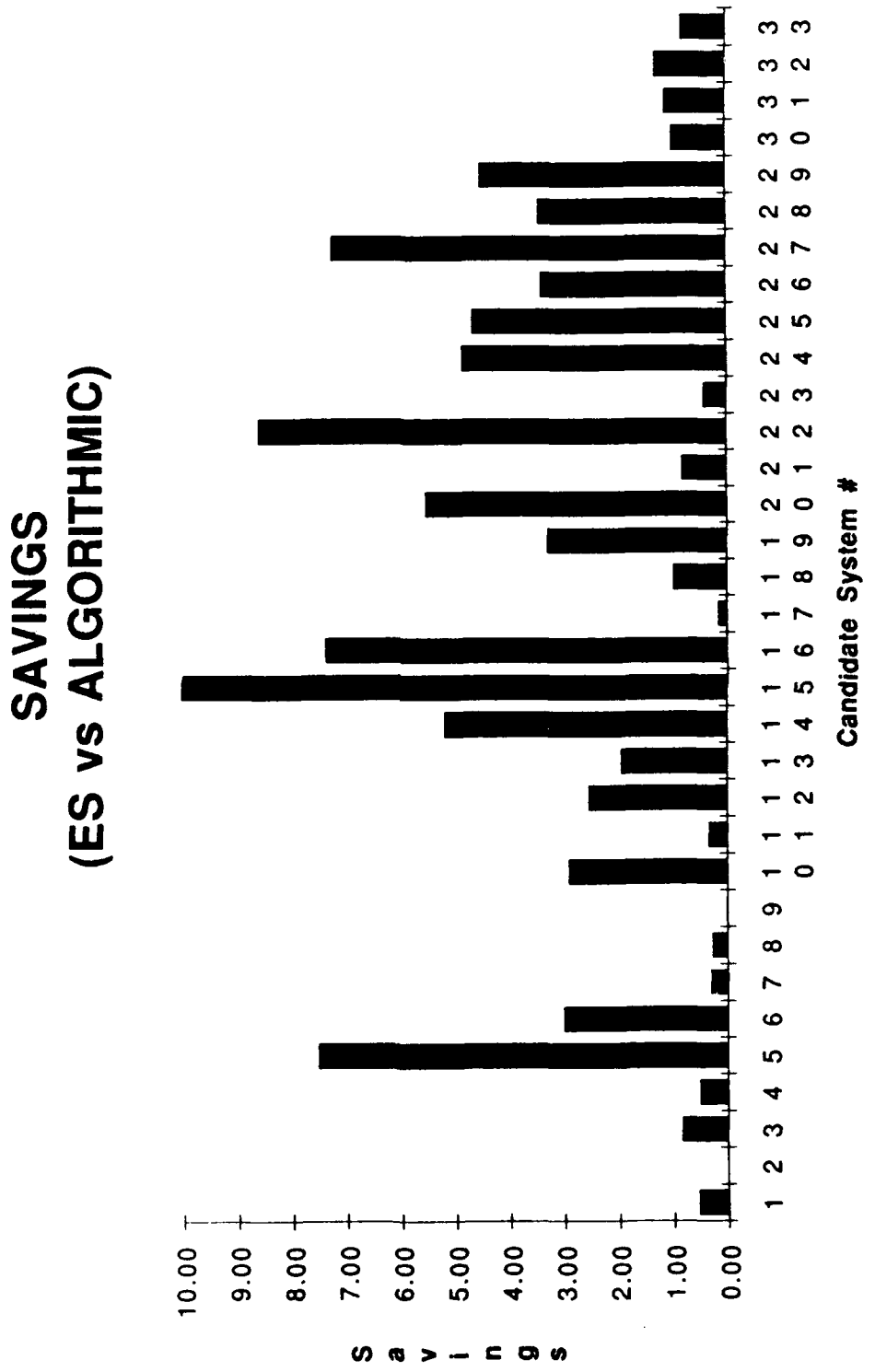


Figure 2.21—Table-5 Cost data intermediate results (Cont.)

TABLE 5.5 SETUP FOR SORTING

	CANDIDATE SYSTEM	SIZE * IMPLE COST (NORM)	SIZE * OPER COST (NORM)	1ST YR EST SAVINGS (NRM)	COMBINED COST (NORM)
1	Data Compression Analysis	0.87	0.18	0.55	0.52
2	Limit Testing	0.06	0.00	0.00	0.03
3	Critical Parameter Vehicle Surveillance	1.08	0.55	0.84	0.82
4	Hazardous Gas Identification and Safein	0.66	0.24	0.51	0.45
5	Operator Training Simulator	8.30	2.95	7.52	5.62
6	Integrated Test Ctlr for Vehicle System	3.38	3.70	3.00	3.54
7	Automatic Remote Sensor Calibration	0.32	0.11	0.30	0.22
8	Automatic Recorder Assignment	0.40	0.23	0.27	0.32
9	Launch Complex Environ. Control System	0.00	0.08	0.01	0.04
10	Operation Troubleshooting	2.87	2.95	2.90	2.91
11	Vehicle Processing Logger System	0.49	0.28	0.34	0.38
12	In-Flight Engine Perform. Monitor	2.87	3.10	2.54	2.98
13	Fluids Analysis Health Monitoring	2.36	2.70	1.94	2.53
14	Abort/Alternative Mission Modes (AGN&	6.18	3.70	5.19	4.94
15	Pre-Flight Test Analysis	10.00	10.00	10.00	10.00
16	Post Flight Telemetry Data Analysis	7.54	6.98	7.37	7.26
17	Flight Control Power Applic. and Monito	0.13	0.11	0.16	0.12
18	Pneumatics, Press., and Purge Controls	1.08	0.53	0.98	0.81
19	Propellant Tanking of Vehicle	3.38	1.74	3.28	2.56
20	Engine Ignition Ground Perform. Mon.	5.92	4.46	5.53	5.19
21	Guidance Calibration	1.08	1.18	0.81	1.13
22	Mission Planning with Automated Navig	7.54	8.74	8.59	8.14
23	Range Safety System	0.49	0.23	0.41	0.36
24	Command and Control Scheduler	4.39	5.97	4.85	5.18
25	Support for the Decision to Launch	4.90	3.70	4.65	4.30
26	System Wide Event Correlation	2.87	5.21	3.39	4.04
27	Vehicle Test Conductor/Scheduler	7.28	6.98	7.23	7.13
28	Facilities Manager	3.38	5.21	3.44	4.29
29	Mission Design Automation	3.97	4.33	4.51	4.15
30	Range Safety Sys. and Recovery Operatic	1.08	0.43	0.98	0.76
31	Payload Manifesting	1.08	1.18	1.10	1.13
32	Countdown Operations System Monitor	1.51	0.55	1.30	1.03
33	Telemetry/Landlines Checks & Assignm	1.08	0.18	0.82	0.63

Figure 2.22—Table-5.5 Cost Macro Sorted Data

TABLE 5.5 Sorted (part a)

	CANDIDATE SYSTEM	SIZE * IMPLE COST (NORM)
15	Pre-Flight Test Analysis	10.00
5	Operator Training Simulator	8.30
16	Post Flight Telemetry Data Analysis	7.54
22	Mission Planning with Automated Navig	7.54
27	Vehicle Test Conductor/Scheduler	7.28
14	Abort/Alternative Mission Modes (AGN&	6.18
20	Engine Ignition Ground Perform. Mon.	5.92
25	Support for the Decision to Launch	4.90
24	Command and Control Scheduler	4.39
29	Mission Design Automation	3.97
6	Integrated Test Ctlr for Vehicle System	3.38
19	Propellant Tanking of Vehicle	3.38
28	Facilities Manager	3.38
10	Operation Troubleshooting	2.87
12	In-Flight Engine Perform. Monitor	2.87
26	System Wide Event Correlation	2.87
13	Fluids Analysis Health Monitoring	2.36
32	Countdown Operations System Monitor	1.51
3	Critical Parameter Vehicle Surveillance	1.08
18	Pneumatics, Press., and Purge Controls	1.08
21	Guidance Calibration	1.08
30	Range Safety Sys. and Recovery Operatic	1.08
31	Payload Manifesting	1.08
33	Telemetry/Landlines Checks & Assignm	1.08
1	Data Compression Analysis	0.87
4	Hazardous Gas Identification and Safein	0.66
11	Vehicle Processing Logger System	0.49
23	Range Safety System	0.49
8	Automatic Recorder Assignment	0.40
7	Automatic Remote Sensor Calibration	0.32
17	Flight Control Power Applic. and Monito	0.13
2	Limit Testing	0.06
9	Launch Complex Environ. Control System	0.00

	CANDIDATE SYSTEM	SIZE * OPER COST (NORM)
15	Pre-Flight Test Analysis	10.00
22	Mission Planning with Automated Navig	8.74
16	Post Flight Telemetry Data Analysis	6.98
27	Vehicle Test Conductor/Scheduler	6.98
24	Command and Control Scheduler	5.97
26	System Wide Event Correlation	5.21
28	Facilities Manager	5.21
20	Engine Ignition Ground Perform. Mon.	4.46
29	Mission Design Automation	4.33
6	Integrated Test Ctlr for Vehicle System	3.70
14	Abort/Alternative Mission Modes (AGN&	3.70
25	Support for the Decision to Launch	3.70
12	In-Flight Engine Perform. Monitor	3.10
5	Operator Training Simulator	2.95
10	Operation Troubleshooting	2.95
13	Fluids Analysis Health Monitoring	2.70
19	Propellant Tanking of Vehicle	1.74
21	Guidance Calibration	1.18
31	Payload Manifesting	1.18
3	Critical Parameter Vehicle Surveillance	0.55
32	Countdown Operations System Monitor	0.55
18	Pneumatics, Press., and Purge Controls	0.53
30	Range Safety Sys. and Recovery Operatic	0.43
11	Vehicle Processing Logger System	0.28
4	Hazardous Gas Identification and Safein	0.24
8	Automatic Recorder Assignment	0.23
23	Range Safety System	0.23
1	Data Compression Analysis	0.18
33	Telemetry/Landlines Checks & Assignm	0.18
7	Automatic Remote Sensor Calibration	0.11
17	Flight Control Power Applic. and Monito	0.11
9	Launch Complex Environ. Control System	0.08
2	Limit Testing	0.00

Figure 2.22—Table-5.5 Cost Macro Sorted Data (Cont.)

TABLE 5.5 Sorted (part b)

	CANDIDATE SYSTEM	1ST YR EST SAVINGS (NRM)
15	Pre-Flight Test Analysis	10.00
22	Mission Planning with Automated Navig	8.59
5	Operator Training Simulator	7.52
16	Post Flight Telemetry Data Analysis	7.37
27	Vehicle Test Conductor/Scheduler	7.23
20	Engine Ignition Ground Perform. Mon.	5.53
14	Abort/Alternative Mission Modes (AGN&	5.19
24	Command and Control Scheduler	4.85
25	Support for the Decision to Launch	4.65
29	Mission Design Automation	4.51
28	Facilities Manager	3.44
26	System Wide Event Correlation	3.39
19	Propellant Tanking of Vehicle	3.28
6	Integrated Test Ctlr for Vehicle System	3.00
10	Operation Troubleshooting	2.90
12	In-Flight Engine Perform. Monitor	2.54
13	Fluids Analysis Health Monitoring	1.94
32	Countdown Operations System Monitor	1.30
31	Payload Manifesting	1.10
18	Pneumatics, Press., and Purge Controls	0.98
30	Range Safety Sys. and Recovery Operatic	0.98
3	Critical Parameter Vehicle Surveillance	0.84
33	Telemetry/Landlines Checks & Assignm	0.82
21	Guidance Calibration	0.81
1	Data Compression Analysis	0.55
4	Hazardous Gas Identification and Safein	0.51
23	Range Safety System	0.41
11	Vehicle Processing Logger System	0.34
7	Automatic Remote Sensor Calibration	0.30
8	Automatic Recorder Assignment	0.27
17	Flight Control Power Applic. and Monitor	0.16
9	Launch Complex Environ. Control System	0.01
2	Limit Testing	0.00

	CANDIDATE SYSTEM	COMBINED COST (NORM)
15	Pre-Flight Test Analysis	10.00
22	Mission Planning with Automated Navig	8.14
16	Post Flight Telemetry Data Analysis	7.26
27	Vehicle Test Conductor/Scheduler	7.13
5	Operator Training Simulator	5.62
20	Engine Ignition Ground Perform. Mon.	5.19
24	Command and Control Scheduler	5.18
14	Abort/Alternative Mission Modes (AGN&	4.94
25	Support for the Decision to Launch	4.30
28	Facilities Manager	4.29
29	Mission Design Automation	4.15
26	System Wide Event Correlation	4.04
6	Integrated Test Ctlr for Vehicle System	3.54
12	In-Flight Engine Perform. Monitor	2.98
10	Operation Troubleshooting	2.91
19	Propellant Tanking of Vehicle	2.56
13	Fluids Analysis Health Monitoring	2.53
21	Guidance Calibration	1.13
31	Payload Manifesting	1.13
32	Countdown Operations System Monitor	1.03
3	Critical Parameter Vehicle Surveillance	0.82
18	Pneumatics, Press., and Purge Controls	0.81
30	Range Safety Sys. and Recovery Operatic	0.76
33	Telemetry/Landlines Checks & Assignm	0.63
1	Data Compression Analysis	0.52
4	Hazardous Gas Identification and Safein	0.45
11	Vehicle Processing Logger System	0.38
23	Range Safety System	0.36
8	Automatic Recorder Assignment	0.32
7	Automatic Remote Sensor Calibration	0.22
17	Flight Control Power Applic. and Monitor	0.12
9	Launch Complex Environ. Control System	0.04
2	Limit Testing	0.03

Figure 2.22—Table-5.5 Cost Macro Sorted Data (Cont.)

- **About Table 6**

Table 6 is the final results or data on which the arbitrator is to base the expert system selection upon. The results in Table 6 are a set of data that is generated from Table 0 and Table 4. The results that are generated are necessary for the arbitrator to make the final decision on which of the candidates would be generated as an expert system and in which order the candidates should be generated as an expert system. The tables that are generated in this document are only a set of supporting data as to why to make a decision in selecting an application to be generated as an expert system, and not as a device to actually make the decision as to which candidate system should be generated as an expert system. The candidate's tabulations of Table 6 are then plotted in the same fashion as they were plotted in Table 4 and Table 5. A list of the charts for Table 6 are listed below. The conclusions of Table 6 are further processed in Table 6.5 into a sorted list of the candidates showing the candidates with the most desirable result of the evaluations as the largest values and the least desirable results as the smallest values. This additional table allows one to see at a glance the candidate's location in that particular list or column of the table.

- **Table 6 charts**

  KBS SUITABILITY

  IMPLEMENTATION RISK AVOIDANCE

  OPERATIONAL COST IMPROVEMENT

  TURN-AROUND TIME REDUCTION

  SAFETY IMPROVEMENT

  FINAL ASSESSMENT (=summation of all other values in Table 6)

- **Table 6 equations**

KBS Suitability	= Table 4(KBS Suitability)
Implementation Risk Avoidance	= -1 * Table 4(Implementation Risk)
Operational Cost Improvement	= (Table 0(Operational Cost No Automation) - Table 0(Oper. LOC/Yr with ES)) * Table 0(Expected Lifetime) - Table 0(Implementation LOC with ES)
Turn around Time Reduction	= Table 0(Schedule Savings if Automated)
Safety Improvement	= Table 0(Danger Level if not Automated) - Table 4(Operational Risk) * Table 0(Mission-Critical Measurement)

TABLE 6 (part a)

	CANDIDATE SYSTEM	KBS Suitability	Implementation Risk Avoidance	Operational Cost Improvement	Turnaround Time Reduction	Safety Improvement
1	Data Compression Analysis	-48	-15	200.00	0	-120
2	Limit Testing	-31	46	400.00	0	99
3	Critical Parameter Vehicle Surveillance	-11	26	2,200.00	0	99
4	Hazardous Gas Identification and Safein	-11	26	1,500.00	0	-90
5	Operator Training Simulator	33	4	13,000.00	0	36
6	Integrated Test Ctlr for Vehicle System	41	4	17,200.00	0	-55
7	Automatic Remote Sensor Calibration	-21	26	500.00	0	33
8	Automatic Recorder Assignment	9	26	900.00	0	22
9	Launch Complex Environ. Control System	9	46	800.00	0	155
10	Operation Troubleshooting	31	46	14,600.00	0	31
11	Vehicle Processing Logger System	49	66	1,000.00	0	51
12	In-Flight Engine Perform. Monitor	-20	-15	14,400.00	0	-300
13	Fluids Analysis Health Monitoring	-20	-15	10,800.00	0	-270
14	Abort/Alternative Mission Modes (AGN&(	-8	-55	15,000.00	0	-300
15	Pre-Flight Test Analysis	31	46	54,000.00	0	204
16	Post Flight Telemetry Data Analysis	31	66	36,800.00	0	102
17	Flight Control Power Applic. and Monito	-21	26	900.00	0	55
18	Pneumatics, Press., and Purge Controls	9	26	2,600.00	0	66
19	Propellant Tanking of Vehicle	9	26	8,600.00	0	66
20	Engine Ignition Ground Perform. Mon.	-11	-14	21,700.00	0	-203
21	Guidance Calibration	-11	26	4,800.00	0	44
22	Mission Planning with Automated Navig	59	66	54,800.00	0	357
23	Range Safety System	-1	26	1,000.00	0	99
24	Command and Control Scheduler	49	26	35,600.00	0	66
25	Support for the Decision to Launch	-8	5	18,500.00	0	60
26	System Wide Event Correlation	60	65	32,600.00	0	90
27	Vehicle Test Conductor/Scheduler	49	26	37,600.00	0	66
28	Facilities Manager	49	86	27,600.00	0	51
29	Mission Design Automation	49	86	27,200.00	0	51
30	Range Safety Sys. and Recovery Operatic	-31	26	2,000.00	0	88
31	Payload Manifesting	49	66	6,800.00	0	31
32	Countdown Operations System Monitor	11	24	1,900.00	0	54
33	Telemetry/Landlines Checks & Assignme	-11	26	200.00	0	55

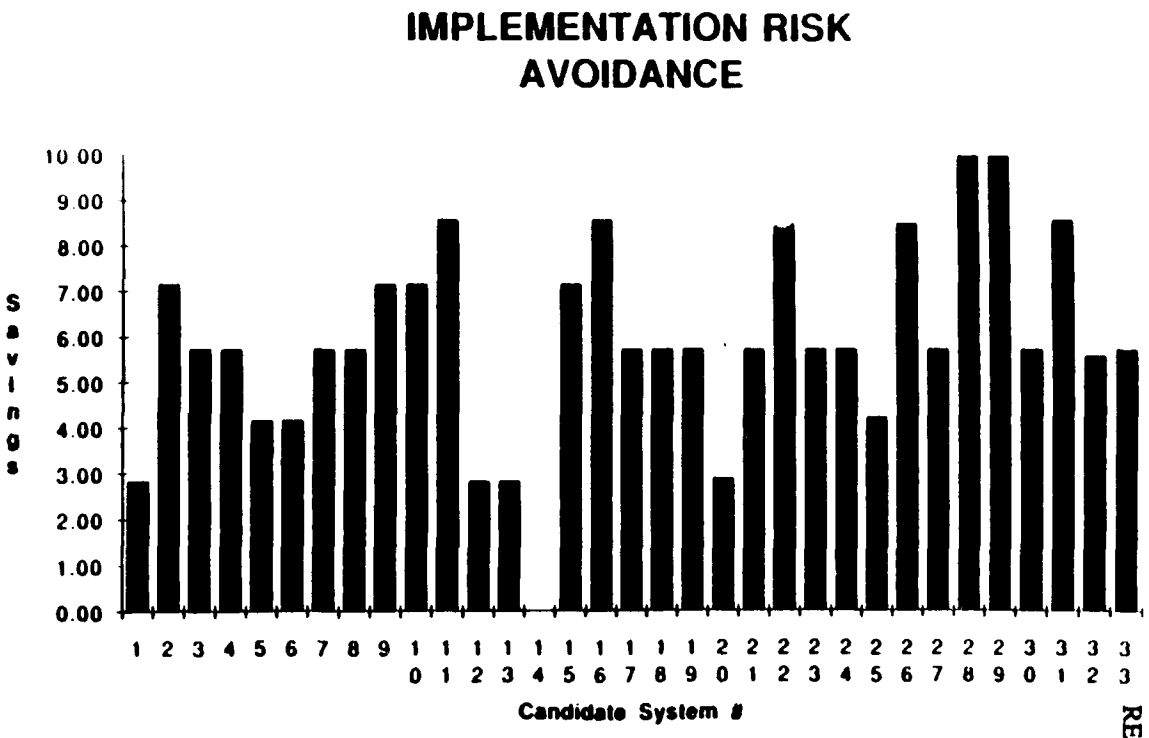
Figure 2.23—Table-6 Domain specific facotrs

TABLE 6 (part b) SCALED

	CANDIDATE SYSTEM	KBS Suitability	Implementation Risk Avoidance	Operational Cost Improvement	Turnaround Time Reduction	Safety Improvement
1	Data Compression Analysis	0.00	2.84	0.00	0.00	2.74
2	Limit Testing	1.57	7.16	0.04	0.00	6.07
3	Critical Parameter Vehicle Surveillance	3.43	5.74	0.37	0.00	6.07
4	Hazardous Gas Identification and Safein	3.43	5.74	0.24	0.00	3.20
5	Operator Training Simulator	7.50	4.18	2.34	0.00	5.11
6	Integrated Test Ctlr for Vehicle System	8.24	4.18	3.11	0.00	3.73
7	Automatic Remote Sensor Calibration	2.50	5.74	0.05	0.00	5.07
8	Automatic Recorder Assignment	5.28	5.74	0.13	0.00	4.90
9	Launch Complex Environ. Control System	5.28	7.16	0.11	0.00	6.93
10	Operation Troubleshooting	7.31	7.16	2.64	0.00	5.04
11	Vehicle Processing Logger System	8.98	8.58	0.15	0.00	5.34
12	In-Flight Engine Perform. Monitor	2.59	2.84	2.60	0.00	0.00
13	Fluids Analysis Health Monitoring	2.59	2.84	1.94	0.00	0.46
14	Abort/Alternative Mission Modes (AGN&	3.70	0.00	2.71	0.00	0.00
15	Pre-Flight Test Analysis	7.31	7.16	9.85	0.00	7.67
16	Post Flight Telemetry Data Analysis	7.31	8.58	6.70	0.00	6.12
17	Flight Control Power Applic. and Monito	2.50	5.74	0.13	0.00	5.40
18	Pneumatics, Press., and Purge Controls	5.28	5.74	0.44	0.00	5.57
19	Propellant Tanking of Vehicle	5.28	5.74	1.54	0.00	5.57
20	Engine Ignition Ground Perform. Mon.	3.43	2.91	3.94	0.00	1.48
21	Guidance Calibration	3.43	5.74	0.84	0.00	5.24
22	Mission Planning with Automated Navig	9.91	8.58	10.00	0.00	10.00
23	Range Safety System	4.35	5.74	0.15	0.00	6.07
24	Command and Control Scheduler	8.98	5.74	6.48	0.00	5.57
25	Support for the Decision to Launch	3.70	4.26	3.35	0.00	5.48
26	System Wide Event Correlation	10.00	8.51	5.93	0.00	5.94
27	Vehicle Test Conductor/Scheduler	8.98	5.74	6.85	0.00	5.57
28	Facilities Manager	8.98	10.00	5.02	0.00	5.34
29	Mission Design Automation	8.98	10.00	4.95	0.00	5.34
30	Range Safety Sys. and Recovery Operatic	1.57	5.74	0.33	0.00	5.91
31	Payload Manifesting	8.98	8.58	1.21	0.00	5.04
32	Countdown Operations System Monitor	5.46	5.60	0.31	0.00	5.39
33	Telemetry/Landlines Checks & Assignm	3.43	5.74	0.00	0.00	5.40

Figure 2.23—Table-6 Domain specific facotrs (sorted)

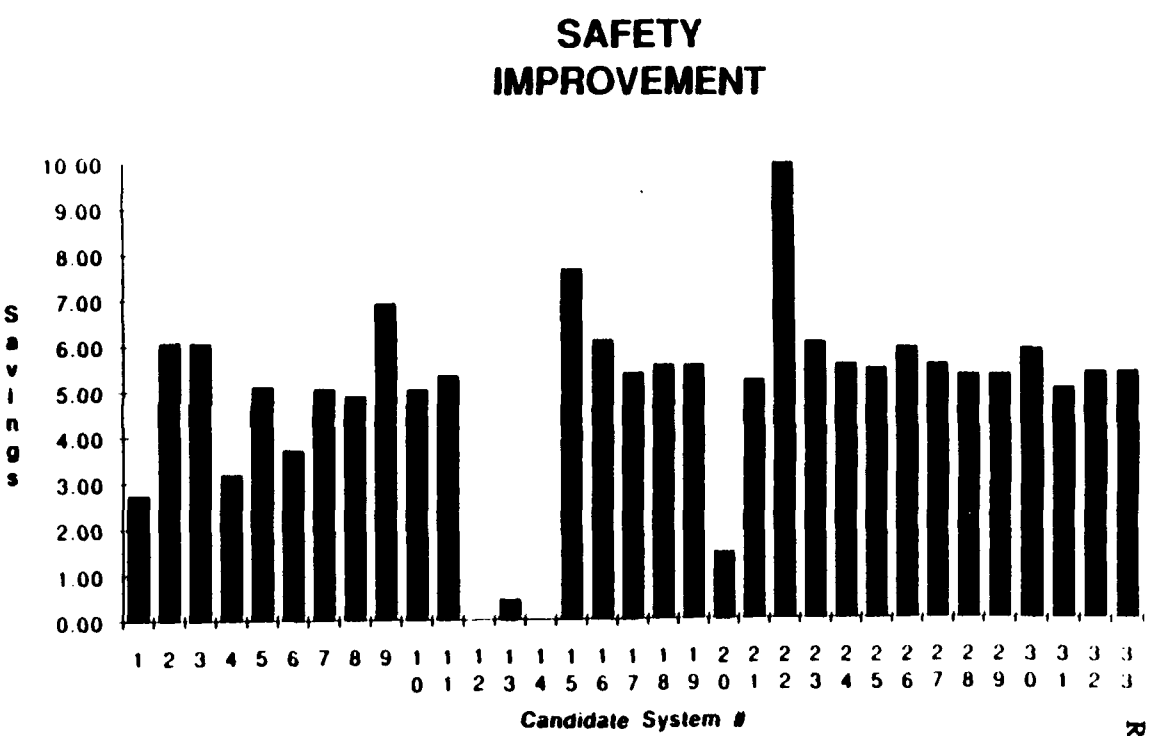
	CANDIDATE SYSTEM	IMPLEM RISK AVOIDANCE
28	Facilities Manager	10 00
29	Mission Design Automation	10 00
11	Vehicle Processing Logger System	8 58
16	Post Flight Telemetry Data Analysis	8 58
22	Mission Planning with Automated Navig	8 58
31	Payload Manifesting	8 58
26	System Wide Event Correlation	8 51
2	Limit Testing	7 16
9	Launch Complex Environ Control System	7 16
10	Operation Troubleshooting	7 16
15	Pre-Flight Test Analysis	7 16
3	Critical Parameter Vehicle Surveillance	5 74
4	Hazardous Gas Identification and Safety	5 74
7	Automatic Remote Sensor Calibration	5 74
8	Automatic Recorder Assignment	5 74
17	Flight Control Power Applic and Monitor	5 74
18	Pneumatics, Press, and Purge Controls	5 74
19	Propellant Tanking of Vehicle	5 74
21	Guidance Calibration	5 74
23	Range Safety System	5 74
24	Command and Control Scheduler	5 74
27	Vehicle Test Conductor/Scheduler	5 74
30	Range Safety Sys and Recovery Operatio	5 74
33	Telemetry/Landlines Checks & Assignm	5 74
32	Countdown Operations System Monitor	5 60
25	Support for the Decision to Launch	4 26
5	Operator Training Simulator	4 18
6	Integrated Test Ctlr for Vehicle System	4 18
20	Engine Ignition Ground Perform Mon	2 91
1	Data Compression Analysis	2 84
12	In Flight Engine Perform Monitor	2 84
13	Fluids Analysis Health Monitoring	2 84
14	Abort/Alternative Mission Modes (AGN&	0 00



The Risk avoidance favors data processing utilities

Figure 2.23—Table-6 Domain specific facotrs (sorted)

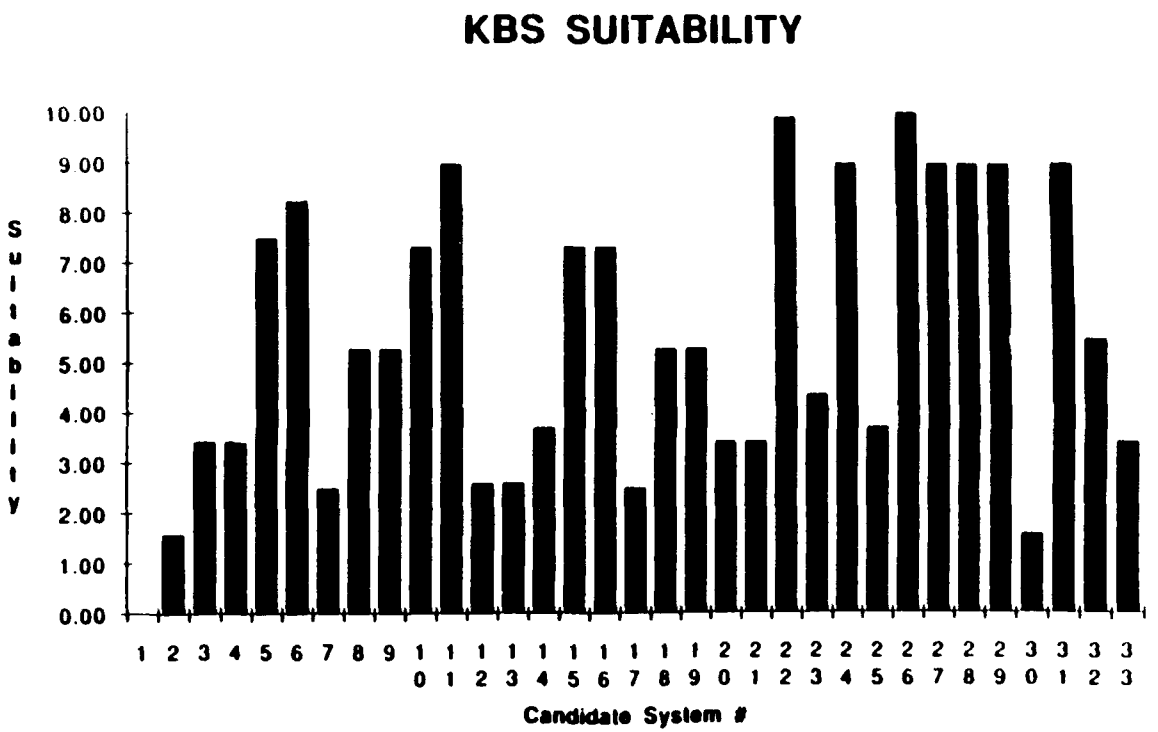
	CANDIDATE SYSTEM	SAFETY IMPROVEMENT
22	Mission Planning with Automated Navig	10 00
15	Pre-Flight Test Analysis	7 67
9	Launch Complex Environ Control System	6 93
16	Post Flight Telemetry Data Analysis	6 12
2	Limit Testing	6 07
3	Critical Parameter Vehicle Surveillance	6 07
23	Range Safety System	6 07
26	System Wide Event Correlation	5 94
30	Range Safety Sys and Recovery Operatic	5 91
18	Pneumatics, Press, and Purge Controls	5 57
19	Propellant Tanking of Vehicle	5 57
24	Command and Control Scheduler	5 57
27	Vehicle Test Conductor/Scheduler	5 57
25	Support for the Decision to Launch	5 48
17	Flight Control Power Applic and Monitor	5 40
33	Telemetry/Landlines Checks & Assignm	5 40
32	Countdown Operations System Monitor	5 39
11	Vehicle Processing Logger System	5 34
28	Facilities Manager	5 34
29	Mission Design Automation	5 34
21	Guidance Calibration	5 24
5	Operator Training Simulator	5 11
7	Automatic Remote Sensor Calibration	5 07
10	Operation Troubleshooting	5 04
31	Payload Manifesting	5 04
8	Automatic Recorder Assignment	4 90
6	Integrated Test Ctlr for Vehicle System	3 73
4	Hazardous Gas Identification and Safein	3 20
1	Data Compression Analysis	2 74
20	Engine Ignition Ground Perform. Mon.	1 48
13	Fluids Analysis Health Monitoring	0 46
12	In-Flight Engine Perform. Monitor	0 00
14	Abort/Alternative Mission Modes (AGN&	0 00



The Safety favors test & data processing utilities

TABLE 4.5 Sorted (part a)

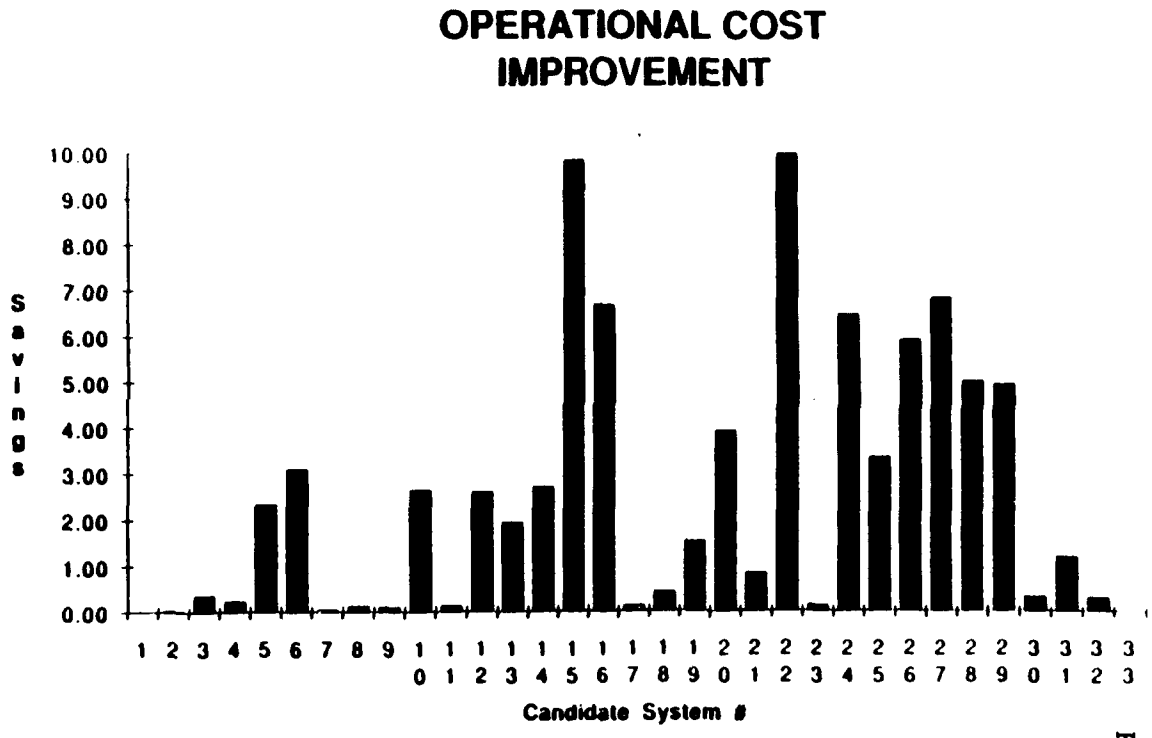
	SORTED CANDIDATE SYSTEMS KBS Suitability	Sorted Actuals
26	System Wide Event Correlation	10 00
22	Mission Planning with Automated Navig	9 91
11	Vehicle Processing Logger System	8 98
24	Command and Control Scheduler	8 98
27	Vehicle Test Conductor/Scheduler	8 98
28	Facilities Manager	8 98
29	Mission Design Automation	8 98
31	Payload Manifesting	8 98
6	Integrated Test Ctlr for Vehicle System	8 24
5	Operator Training Simulator	7 50
10	Operation Troubleshooting	7 31
15	Pre-Flight Test Analysis	7 31
16	Post Flight Telemetry Data Analysis	7 31
32	Countdown Operations System Monitor	5 46
8	Automatic Recorder Assignment	5 28
9	Launch Complex Environ. Control System	5 28
18	Pneumatics, Press., and Purge Controls	5 28
19	Propellant Tanking of Vehicle	5 28
23	Range Safety System	4 35
14	Abort/Alternative Mission Modes (AGN&	3 70
25	Support for the Decision to Launch	3 70
3	Critical Parameter Vehicle Surveillance	3 43
4	Hazardous Gas Identification and Safety	3 43
20	Engine Ignition Ground Perform Mon	3 43
21	Guidance Calibration	3 43
33	Telemetry/Landlines Checks & Assignm	3 43
12	In-Flight Engine Perform Monitor	2 59
13	Fluids Analysis Health Monitoring	2 59
7	Automatic Remote Sensor Calibration	2 50
17	Flight Control Power Applic and Monito	2 50
2	Limit Testing	1 57
30	Range Safety Sys and Recovery Operatic	1 57
1	Data Compression Analysis	0 00



The Suitability factor favors mission operations utilities

Figure 2.23—Table-6 Domain specific facotrs (sorted)

	CANDIDATE SYSTEM	OPER COST IMPROVEMENT
22	Mission Planning with Automated Navig	10 00
15	Pre Flight Test Analysis	9 85
27	Vehicle Test Conductor/Scheduler	6 85
16	Post Flight Telemetry Data Analysis	6 70
24	Command and Control Scheduler	6 48
26	System Wide Event Correlation	5 93
28	Facilities Manager	5 02
29	Mission Design Automation	4 95
20	Engine Ignition Ground Perform Mon	3 94
25	Support for the Decision to Launch	3 35
6	Integrated Test Ctlr for Vehicle System	3 11
14	Abort/Alternative Mission Modes (AGN&	2 71
10	Operation Troubleshooting	2 64
12	In-Flight Engine Perform Monitor	2 60
5	Operator Training Simulator	2 34
13	Fluids Analysis Health Monitoring	1 94
19	Propellant Tanking of Vehicle	1 54
31	Payload Manifesting	1 21
21	Guidance Calibration	0 84
18	Pneumatics, Press , and Purge Controls	0 44
3	Critical Parameter Vehicle Surveillance	0 37
30	Range Safety Sys and Recovery Operatic	0 33
32	Countdown Operations System Monitor	0 31
4	Hazardous Gas Identification and Safety	0 24
11	Vehicle Processing Logger System	0 15
23	Range Safety System	0 15
8	Automatic Recorder Assignment	0 13
17	Flight Control Power Applic and Monitor	0 13
9	Launch Complex Environ Control System	0 11
7	Automatic Remote Sensor Calibration	0 5
2	Limit Testing	0 04
1	Data Compression Analysis	0 00
33	Telemetry/Landlines Checks & Assignm	0 00



The Cost improve. favors launch processing utilities

Figure 2.23—Table-6 Domain specific facotrs (sorted)

- **About 6.5 Macro**

Table 5.5 and Table 6.5 are stored in the same file, separate from the other tables. The macros 5.5 and 6.5 perform the same function for Tables 5.5 and 6.5 that the macro 4.5 does for Table 4.5. In order to sort the candidate list into a list in which the candidates show a ranking of top to bottom values the Excel program requires a macro to be executed. (Example: sort the implementation cost for the candidate list into a list starting with the candidate that would cost the most to implement to the candidate that would cost the least to implement)

Because of the way that the Excel program stores the data after the sort macro has been executed, the sort macro is required to be re-executed every time that the program is loaded, to ensure that the data is sorted into the correct locations for observations or for printing. The sorting macro sorts only the data and not the equations so that when the data is stored back into the file upon closing the file, the data is actually stored into the file back in the table's unsorted state. The program also takes so long to run a recalculation of all the tables every time that a value is changed in any table that the macro changes the Excel program into a manual calculation mode. This means that a manual calculation and/or the sort Macro, has to be executed after any data has been changed or after the program has been started. Otherwise, data in the tables would be inconsistent or incorrect.

[Note: The Excel program operator needs to ensure that the correct table or program is active when the macro is executed or else the macro will walk through a pertinent file sorting the values into an undesirable state]

```
Record1
=CALCULATION(3,FALSE)
=CALCULATE.NOW()
=SELECT("R4C9")
=HLINE(4)
=SELECT("R4C9:R36C11")
=SORT(1,"R4C11",2)
=HLINE(4)
=SELECT("R4C13:R36C15")
=SORT(1,"R4C15",2)
=HLINE(4)
=SELECT("R4C17:R36C19")
=SORT(1,"R4C19",2)
=HLINE(4)
=SELECT("R4C21:R36C23")
=SORT(1,"R4C23",2)
=VSCROLL(0.2258%)
=HSCROLL(0%)
=HLINE(4)
=SELECT("R41C9:R73C11")
=SORT(1,"R41C11",2)
=HLINE(4)
=SELECT("R41C13:R73C15")
=SORT(1,"R41C15",2)
=HLINE(4)
=SELECT("R41C17:R73C19")
=SORT(1,"R41C19",2)
=HLINE(4)
=SELECT("R41C21:R73C23")
=SORT(1,"R41C23",2)
=HLINE(4)
=SELECT("R41C25:R73C27")
=SORT(1,"R41C27",2)
=RETURN()
```

Figure 2-24—Macro for Table 6.5

2.3.4 Assessment Methodology Matrix Summery

- **Table results**

Figure 2-25 lists only the top rated candidates in each of the categories. This list only takes into consideration what the raw data has generated and does not take into account any other reasoning as to why a particular expert system candidate should be chosen and generated as an expert system prior to any of the other posible candidates.

Table 4 Results of the best candidates in each of the sub-categories		
Best KBS Suitability	26	System Wide Event Correlation
Lowest Implementation Cost	11	Vehicle Processing Logger System
Lowest Operational Cost	22	Mission Planning with Automated Navigation
Lowest Implementation Risk	28	Facilities Manager
	29	Mission Design Automation
Lowest Operational Risk	11	Vehicle Processing Logger System
	15	Pre-Flight Test Analysis
	16	Post Flight Telemetry Data Analysis
	22	Mission Planning with Automated Navigation
	28	Facilities Manager
	29	Mission Design Automation
Combined Ranking	22	Mission Planning with Automated Navigation
Table 5 Results of cost impacts		
Smallest Size * Implementation Cost	9	Launch Complex Environ. Control System
Smallest Size * Operational Cost	2	Limit Testing
Largest 1st Yr Est Savings	15	Pre-Flight Test Analysis
Smallest Combined Costs	2	Limit Testing
Table 6 Result of final conclusions		
Best KBS Suitability	26	System Wide Event Correlation
Best Implementation Risk Avoidance	28	Facilities Manager
	29	Mission Design Automation
Best Operational Cost Improvement	22	Mission Planning with Automated Navigation
Turn-around Time Reduction		(Data has not been generated as of yet)
Best Safety Improvement	22	Mission Planning with Automated Navigation
Best Overall Candidate	22	Mission Planning with Automated Navigation

Figure 2-25 Top rated candidates in each of the categories

• Summery of Top Choice CandidatesCandidate selection

The candidates as ranked by the assessment metrics have been divided up into six catagories. These catagories serve as a basis to define the positive features of a Knowledge Based System (KBS) that the engineer should consider while selecting the choice candidates.

- Best Knowledge Based System Suitability
- Lowest Implementation Cost
- Lowest Operational cost
- Lowest Implementation Risk
- Lowest Operational Risk

In implementing the selection process it is recommended that one analyze the candidates by individually catagory rather then by the overall summation of all catagories. This individaul analysis by attribute will permit more flexiability in the evaluation process.

Out of the 33 candidates evaluated six were weighted within the top 70 percent and had a "Final Assessment" Ranking between 27.15 and 38.49.(refer to Table 6.5 Sorted (part c), shown in Figure 2-26). These top leading candidates were:

- **Mission Planning with Automated Navigation [22]** (38.49)
- **Pre-Flight Test Analysis [15]** (32.00)
- **System Wide Event Correlation [26]** (30.38)
- **Facilities Manager [28]** (29.34)
- **Mission Design Automation [29]** (29.27)
- **Post Flight Telemetry Data Analysis [16]** (28.72)
- **Vehicle Test Conductor/Scheduler [27]** (27.15)

The Best Knowledge Based System Suitability

• Recommended for ADP 2102 Prototype Demonstration

The first goal of the ALS Expert Systems Advanced Development Program No. 2302 will be to evaluate the actual cost and performance comparisons of an automated expert system and a non automated conventional system with similar overall requirements. As a ground rule it is imperative that the analysis is based on concrete data from lab and actual field testing.

The second goal of ALS ADP 2302 will be to show that conventional systems can be validated and verified sufficiently to show them worthy of closed loop real-time control of actual applications. Such V&V techniques will need to be applied based on actual known performance criteria from related none ES based systems. It would not be possible to apply fully understood V&V rules and facts without having first hand experience with the related system. It is therefore recommended that the ALS ADP 2102 consider the candidate expert system not solely on the criteria of the ranking order of the candidates (see section 2.3.3.5).

A recommendation is made that the **Propellant Tanking of Vehicle [19]** from the Mission Planning and Operations category be considered from the results of the evaluation for the ALS Phase 2 Smart Ground Service Equipment (GSE) . The Propellant Tanking of Vehicle expert system scored approximately mid-range in the assessment methodology results. Despite not having the highest ranking this candidate was chosen as the "best" option for expert system demonstration. The driving reason behind selection was the engineering design groups familiarization with in-house experience of the existing non-expert system (conventional) based procedures. A large amount of data currently is available for the Atlas and Centaur vehicles propellant tanking conventional systems, and this data will be required to evaluate performance comparisons to the expert system based technology. Most of the other candidate expert systems have never been implemented in any conventionally automated system and it therefore would be difficult to produce performance comparisons based on concrete data.

The tabular assessment methodology yields a non-biased analysis of the different candidate systems. Although this assessment is very valuable as method of ranking candidates it should be regarded as a tool that needs maintenance and tailoring to specific overall requirements. In the assessment analysis many of the candidates that were ranked had no known data for not automated (conventional) verses automated (KBS). Since many of these candidates have never been implemented before there was incomplete data for Table 0 and columns titled "KBS Category", "KBS Example", "Danger Level of Not Automated", and "Schedule Savings if Automated" were left blank. With the absence of this information the analysis is weakened when addressing the direct comparison of existing conventional vs. KBS.

For the analysis the following question with regards to user experience was not directly asked:

- For evaluation and implementation of validation and verification (V&V) techniques does the conventional system have a established historical data base?

This question is critical in the selection of a candidate to support the ADP 2302 task to develop and demonstrate V&V techniques. To evaluate V&V techniques upon a KBS it is imperative that a similar conventional system with related performance parameters be fully understood. To prove the V&V techniques for this one-time ADP 2302 task it imperative for the selection engineering design group to address this above question. It was decided to not include this question in the standard assessment methodology because it does not normally have an impact to the assessment process.

It is important that an evaluation team be aware of the overall situational requirements. This assessment methodology must be used as a tool, with the final decision tempered by experience, programmatic requirements, and economic factors.

	CANDIDATE SYSTEM	Final Assessment
22	Mission Planning with Automated Navig	38 49
15	Pre Flight Test Analysis	32 00
26	System Wide Event Correlation	30 38
28	Facilities Manager	29 34
29	Mission Design Automation	29 27
16	Post Flight Telemetry Data Analysis	28 72
● 27	Vehicle Test Conductor/Scheduler	27 15
24	Command and Control Scheduler	26 78
31	Payload Manifesting	23 81
11	Vehicle Processing Logger System	23 05
10	Operation Troubleshooting	22 15
● 9	Launch Complex Environ Control System	19 48
6	Integrated Test Ctlr for Vehicle System	19 27
5	Operator Training Simulator	19 14
● 19	Propellant Tanking of Vehicle	18 13
18	Pneumatics, Press , and Purge Controls	17 03
25	Support for the Decision to Launch	16 79
32	Countdown Operations System Monitor	16 77
23	Range Safety System	16 32
8	Automatic Recorder Assignment	16 05
3	Critical Parameter Vehicle Surveillance	15 61
21	Guidance Calibration	15 25
2	Limit Testing	14 85
33	Telemetry/Landlines Checks & Assignm	14 57
17	Flight Control Power Applic and Monito	13 78
30	Range Safety Sys and Recovery Operatic	13 55
7	Automatic Remote Sensor Calibration	13 37
4	Hazardous Gas Identification and Salem	12 61
20	Engine Ignition Ground Perform Mon	11 75
12	In Flight Engine Perform Monitor	8 01
13	Fluids Analysis Health Monitoring	7 83
14	Abort/Alternative Mission Modes (AGN&	6 41
1	Data Compression Analysis	5 58

Figure 2-26—Composit Candidate Assessment

SAF — Safety Improvement
TRN — Turn-around Time Reduction
OP — Operational Cost Improvement
IMP — Implementation Risk Avoidance
KBS — Knowledge-base Suitability

FINAL ASSESSMENT
TABLE 6

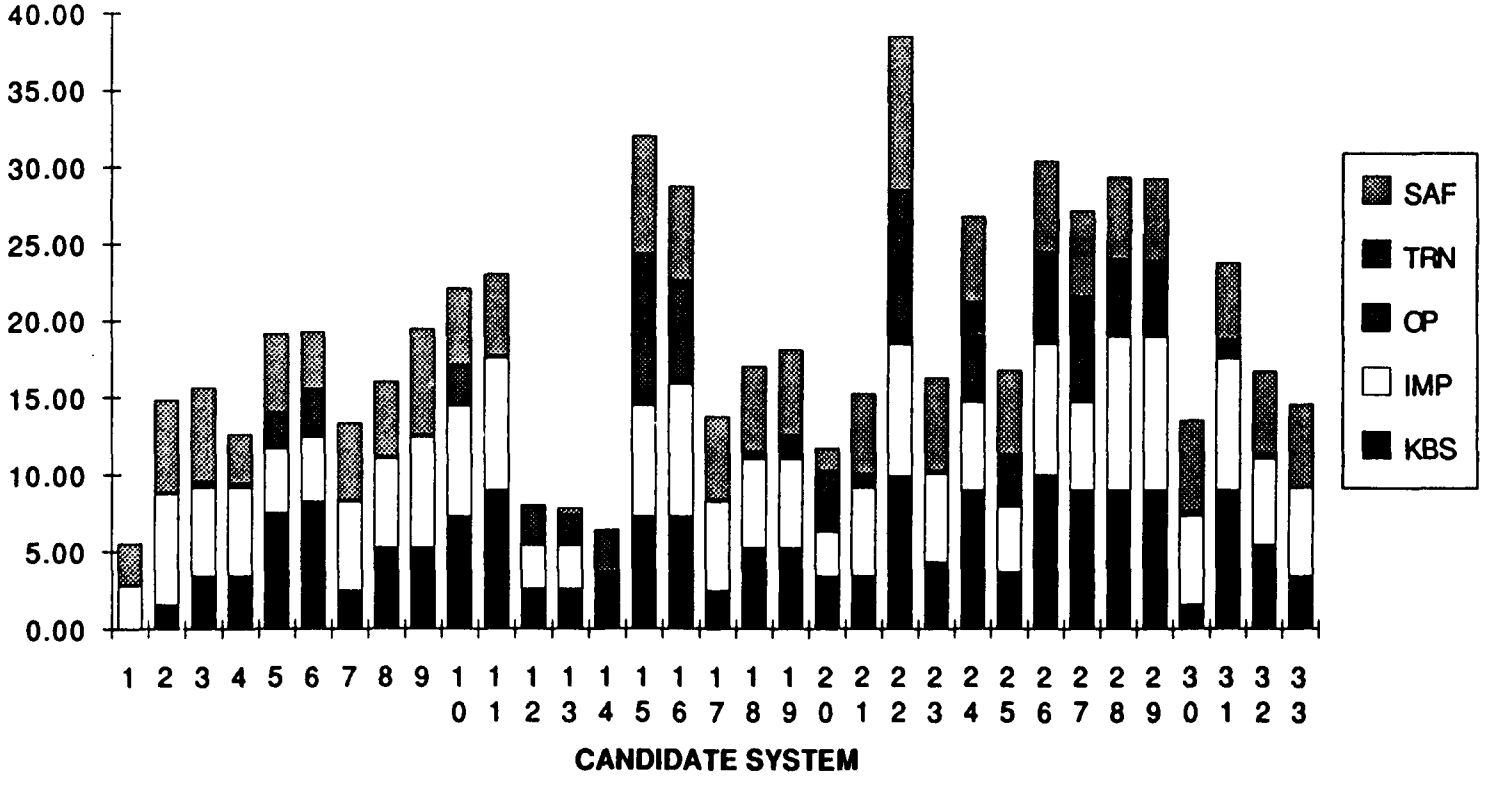


Figure 2.26—Table-6 Composite Candidate Assessment

3. MAXIMUM AUTONOMOUS SYSTEM ARCHITECTURE

ALS requires operational effective and cost benefitial automation, see Sec. 1.5.3. This section discusses implications of maximum autonomy, delineates trade issues in selecting an architecture, proposes an architecture based on the Knowledge bus described in Section 4, and enumerates the benefits of implementing such a maximum autonomous system architecture.

3.1 CONCEPTS

An autonomous system is one which operates free from outside control, with reserved intervention by management. Autonomous systems have some degree of self-determination as to what operations they should perform. For example, an autonomous automobile would run without the need for the driver to steer or press the accelerator pedal. It could be made even more autonomous if it went to fill its gas tank without outside intervention whenever it detected that the fuel had fallen below a certain level. Autonomous systems monitor their environments while carrying out routine operations. If anomalies are detected in the monitored inputs, the system will diagnose the cause of the anomaly and try to correct or compensate in order to continue to operate without intervention. Autonomy does not imply that systems be able to correct unforeseen errors. However, if the cause of an anomaly can be diagnosed as an error which was anticipated, autonomous systems should be able to continue without intervention. Many autonomous systems have the ability to diagnose anomalies caused by faulty equipment and correct for them by repairing the fault or using backup equipment. The use of autonomous systems in a large operation like ALS can reduce or eliminate the need for human intervention for a number of applications.

An analogy can be made between autonomous systems and goal driven (i.e. backward chaining) expert systems. Instead of just being given a predefined set of instructions to carry out its tasks, an autonomous system can be thought of as having a goal that it can achieve using various methods. The system develops a plan to achieve its goal. If the system encounters problems, such as faulty equipment, which cause the plan to fail, it tries a new plan. However, this does not mean that the best way to implement autonomous systems is with expert systems. As we seen in Sec. 2, some autonomous operations are best implemented conventionally while others are well suited for implementation as expert systems.

3.1.1 Examples of Autonomy

Deep space probes must be able to function for days or even weeks without input from earth based ground control, so for many years they have been designed with a high degree of autonomy. Much of their autonomy involves navigation. These spacecraft must constantly monitor their position and plan and execute maneuvers to keep on a desired course. Fighter aircraft and cruise missiles have a similar function. They must be able to detect threats and plan and execute maneuvers to avoid the threats while continuing travel toward a desired target.

Many autonomous systems have implemented some degree of fault tolerance. These systems monitor the status of the hardware for anomalies (fault detection), diagnose the cause of the anomaly (fault isolation), and then formulate and execute a plan for correcting the fault (fault recovery). Figures 3.1 through 3.6 show the data flow for a fault tolerant autonomous control system. A fault tolerant autonomous system takes specifications consisting of goals, guidelines, hardware configuration, and reporting requirements. It then develops a plan for achieving the goals using the current configuration and the guidelines. The autonomous system executes the plan by sending appropriate control commands to the hardware and monitoring sensor data from the hardware. If anomalies are detected in the sensor data, the autonomous system invokes its anomaly resolution component to isolate the fault and determine a new configuration which bypasses the faulty hardware. The new configuration is fed back into the planning process. In this example, if the autonomous system is unable to correct a fault, it issues a demand for human intervention.

3.1.2 Maximum Autonomy for ALS

A maximally autonomous ALS could be imagined where, after input of the specifications for a launch, the entire process of mission planning, vehicle manufacture and assembly, cargo integration, and launch operations would proceed without any human intervention. While such a system would not be technically feasible or cost effective at the present time, it is valuable to consider the systems synergy. Many operations of ALS can and will be automated. Some of these operations could be made autonomous by having them monitor the status of other operations on which they depend and modify their execution when appropriate. Since the ALS design calls for recording of all program information in the ALS Unified Information System (Unis), this status information can be easily accessed.

With this approach, instead of needing human intervention hundreds of times during an ALS mission preparation, the mission can be controlled with human intervention required only in limited strategic places.

While opportunities for autonomous operations in the ALS operations and information segments should not be overlooked, this report concentrates on an autonomous architecture for the ALS vehicle segment. Maximum autonomy can only be achieved by closely coupling ground operations with an autonomous vehicle system, so the architecture for an autonomous ALS vehicle must provide links to Unis and ground operations. An autonomous vehicle would get its primary inputs from the operations segment and, therefore, must be able to monitor the status of operations both directly and by accessing the Unis database. By monitoring ground operations status, a maximally autonomous vehicle could perform the following functions with minimal human intervention:

- Adaptive Guidance and Navigation
- Flight Control and Sequencing
- Anomaly Recovery (Avionics & Engines)
- Booster Recovery (PA Module)
- Low-Level Hardware Interface
- Vehicle Heath Monitoring
- Data Communications

3.2 ANALYSIS

3.2.1 Implications of Maximum Autonomy

3.2.1.1 Maximum Available Data

In order to achieve maximum autonomy, a system must be able to access all of the data necessary to analyze anomalies and develop plans to meet desired goals. Data can come from on-board sensors, from the ground, and from other sub-systems. An autonomous system may warrent try out its plans using a simulation model if time allows. Autonomous systems which make decisions in place of human beings must have access to the same knowledge that the human decision makers have. However, not all human knowledge can be captured in a form that an automated system can use, so some decision making will still be need to be done by humans. In some cases, even with incomplete knowledge, an autonomous system may perform better than a human since it would be less prone to make errors due to fatigue and it can analyze much more data in a short time. In areas where human decisions are still made, an autonomous system can still be used to verify that input directions are reasonable as a decision aid.

3.2.1.2 Flexible Distributed Functional Allocation

An autonomous ALS vehicle segment needs to access information from the operations and information segments. This information includes mission goals and system configuration. Data from the ground can supplement the data available from on-board sensors. Human intervention will also be input via the operations segment. Maximum autonomy will entail both an on-board vehicle mission management system (VMMS) and related ground systems. Since ALS is a family of vehicles, the exact nature of the data needed from the operations segment is likely to vary. A maximum autonomous system architecture must allow a flexible allocation of functions between on-board and ground computers. The architecture should provide a flexible interface through which various kinds of data can be shared between an on-board VMMS and the ground.

The use of a distributed computing environment has many benefits. Maximum autonomy involves many operations which must be performed under stringent time constraints. Current technology may not provide the cost effective computing power to meet these constraints in a single processor environment, however, these operations can be performed by distributing them across multiple computing platforms, suited to each application and more easily up gradable over the life cyle. Additionally, a distributed system can be used to achieve some degree of fault tolerance. By having redundant operations on multiple platforms, if one platform should go down the system can continue to function. A voting mechanism can be used to determine if one of the machines is malfunctioning.

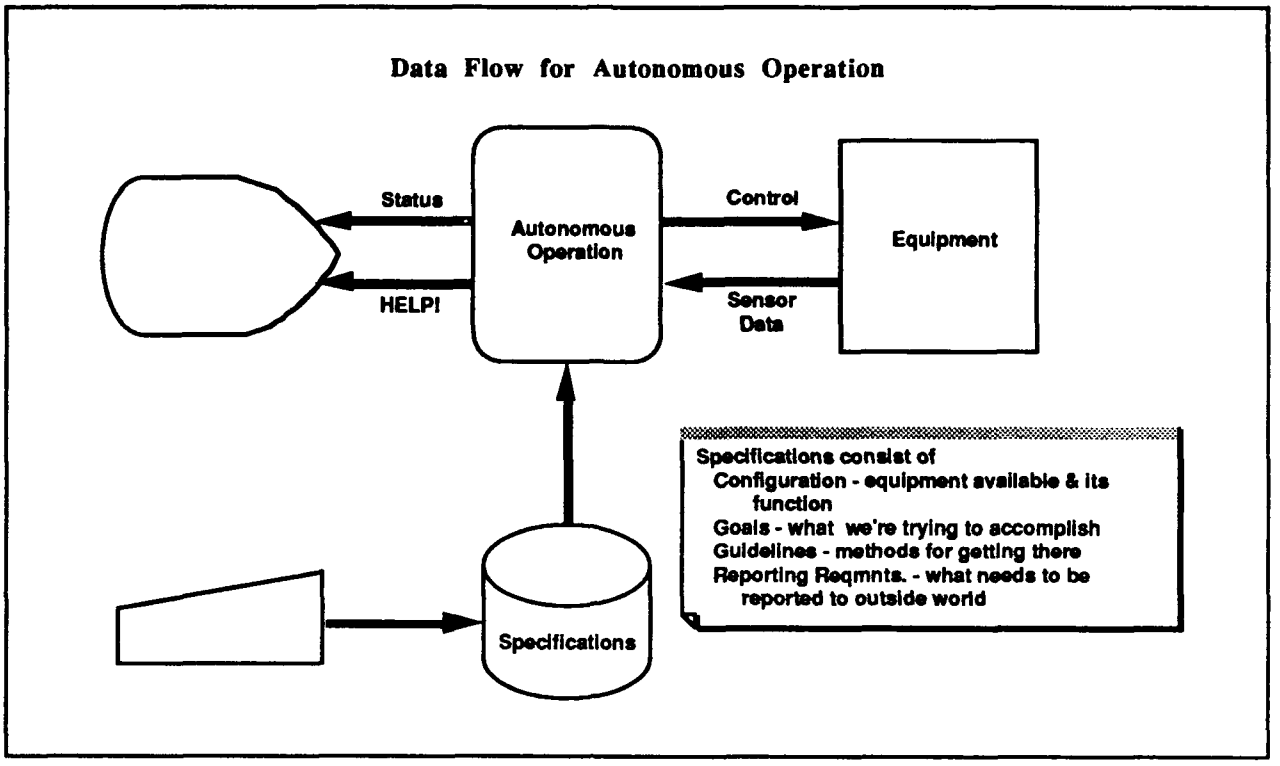


Figure 3.1—A fault tolerant autonomous system takes specifications consisting of goals, guidelines, configuration, and reporting requirements; controls equipment while monitoring sensor data for anomalies; and switches to a new configuration when it detects faulty hardware

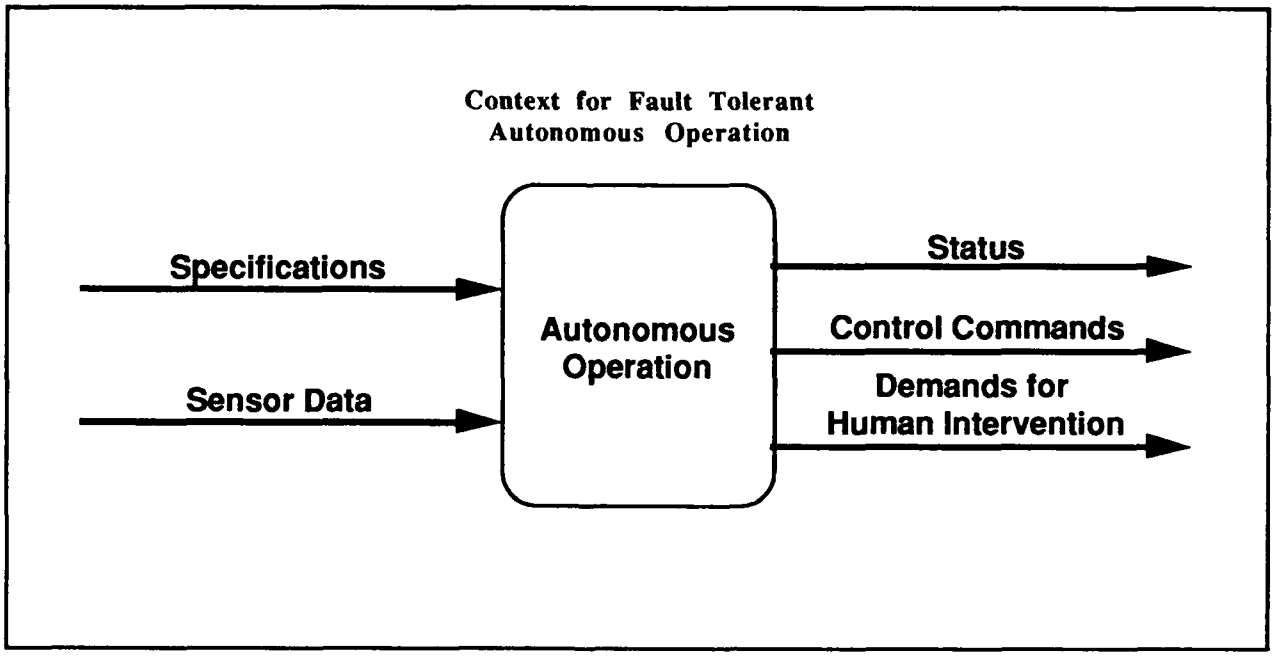


Figure 3.2—The Context for Autonomous Operation—demands for human intervention are made when the system cannot correct the fault by itself

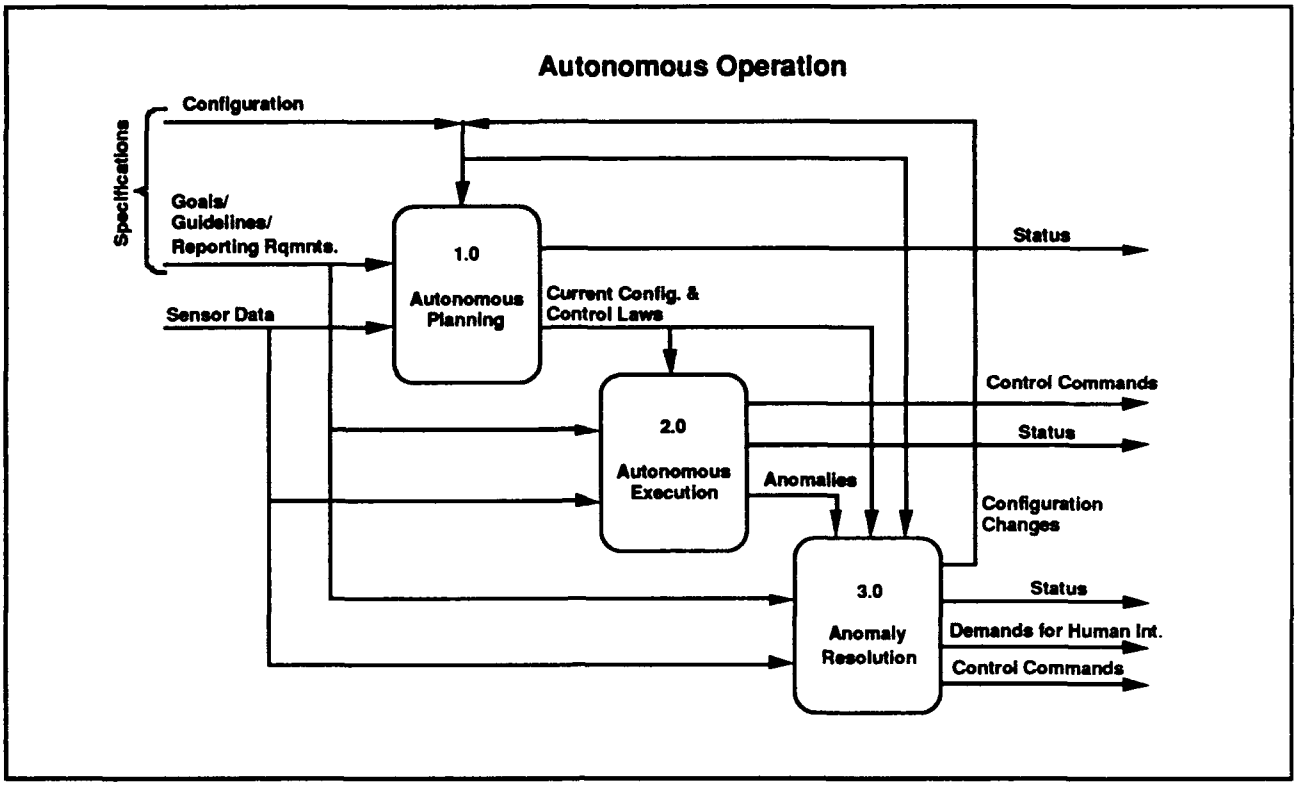


Figure 3.3—Autonomous Operation develops plans for achieving goals, executes the plans, and resolves anomalies encountered during execution

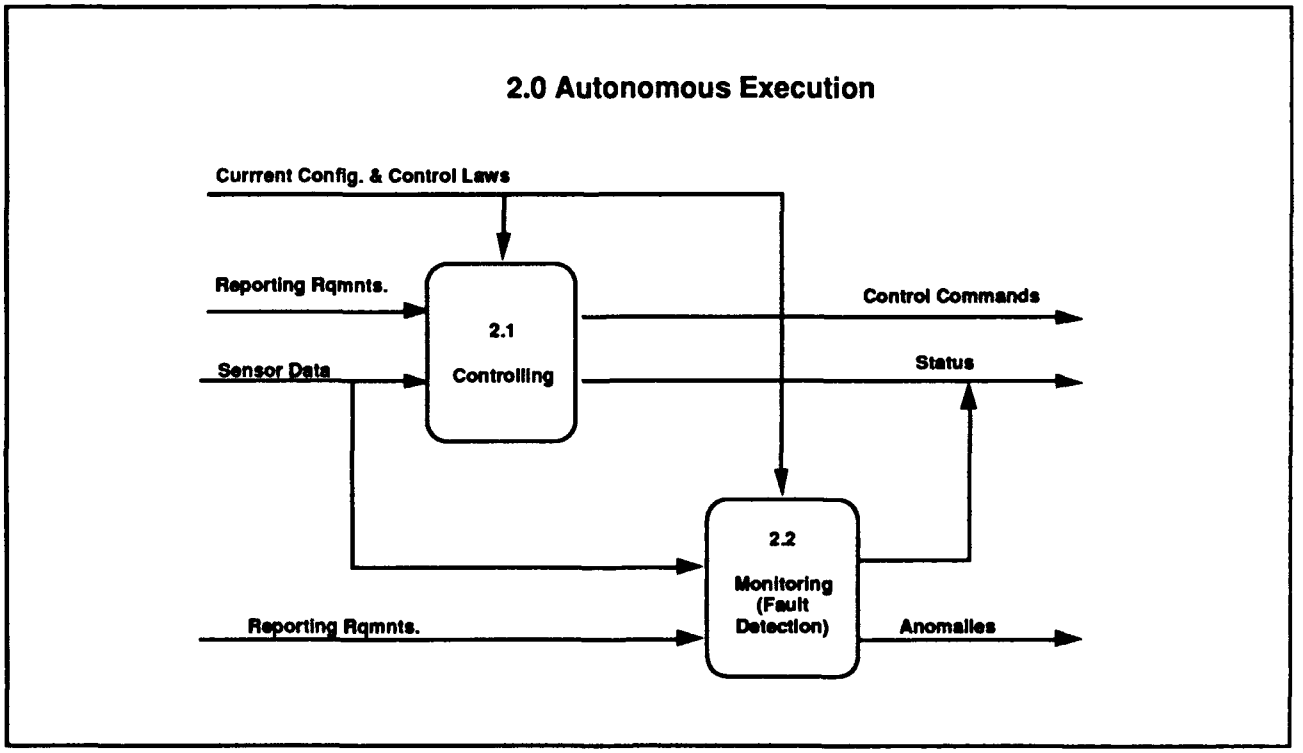


Figure 3.4—Autonomous Execution controls the hardware and monitors sensor data for anomalies

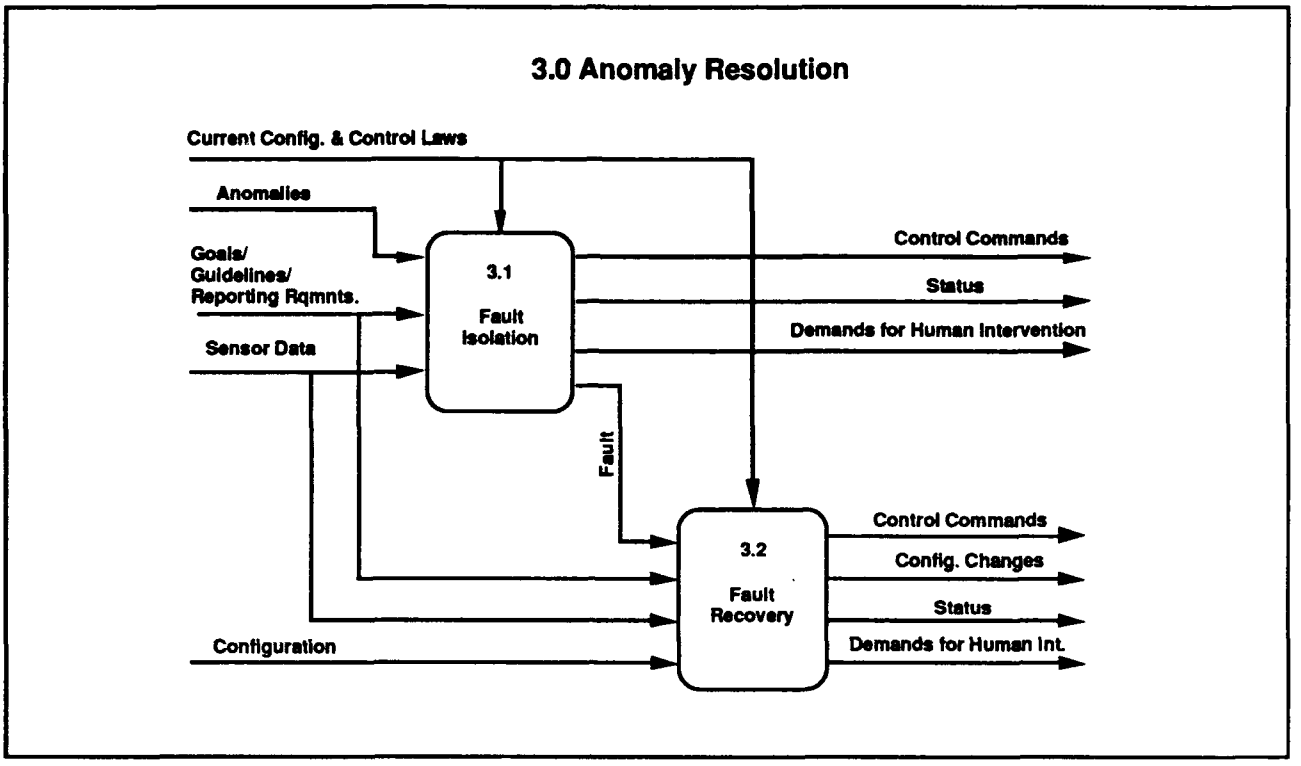


Figure 3.5—Anomaly Resolution consists of isolating the fault and finding a way to bypass the faulty hardware

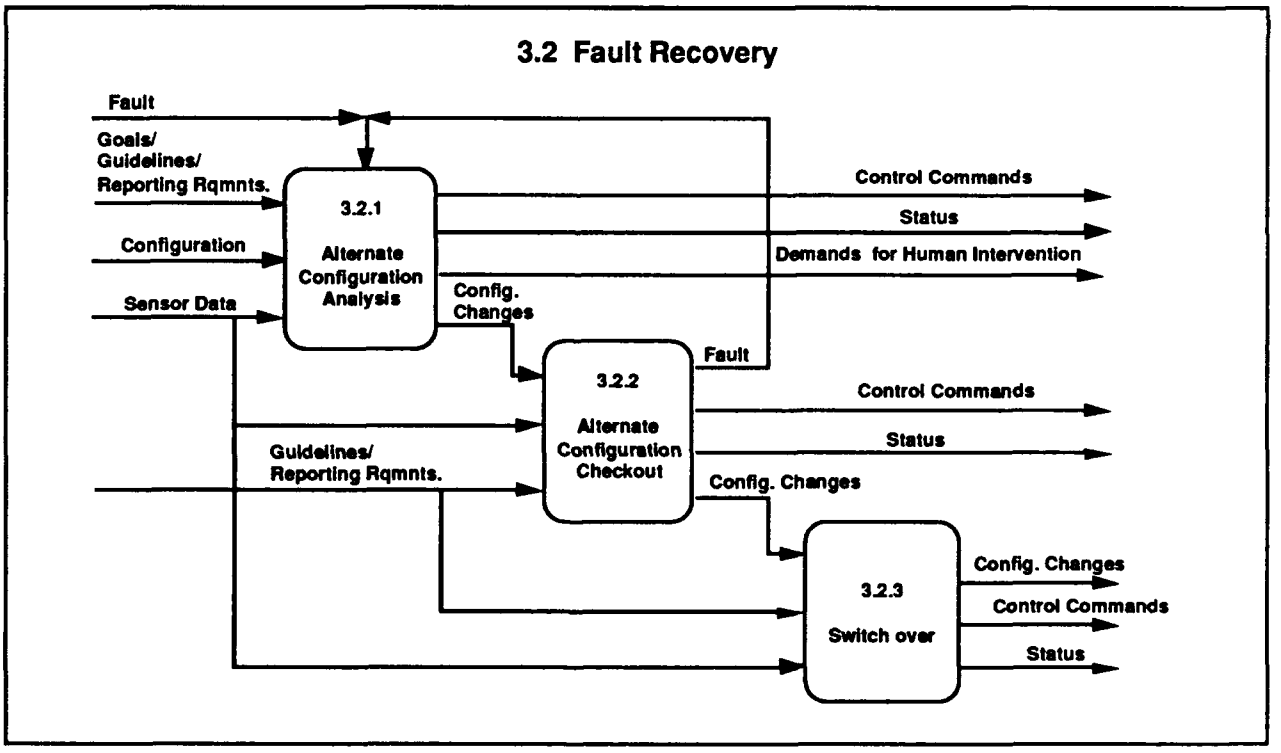


Figure 3.6—Fault Recovery determines a new configuration which bypasses the faulty hardware

Distributed systems do have recognized problems that must be avoided. Distributed system must be designed to support the necessary interchange of data between platforms while avoiding communication bottlenecks. Care must be taken in the design to avoid propagation of errors from one platform to others. A properly designed distributed architecture will support flexible allocation of functions by making the location transparent to others through the use of a global name server.

3.2.1.3 Technology Transparency

Over the lifetime of ALS it is expected that new technologies will be incorporated into the system if they result in cost savings and improved operability. These technological advances may include new information and knowledge processing techniques as well as advances in spacecraft hardware and operations. Operations which previously were not automated may become candidates for autonomous operation, and previously automated operations may be improved by using new technology. The architecture for the autonomous system should be designed so that it can accommodate new technologies when they become available.

3.2.1.4 Verification and Validation

The verification and validation of an autonomous system, especially one using knowledge based methods, present a new set of problems which need to be addressed. Autonomous operations must be made very reliable since they will be making decisions in place of humans. Even with override capabilities, autonomous systems must be able to react to many different situations. The specifications for an autonomous system must be carefully verified to see that they are complete and that they correctly define the actions to be taken in all situations. The design and implementation must be verified to assure that the specifications are met whether the system is implemented conventionally or with an expert system. The issue of validating the system is challenging since it will be hard to test all the situations which might be encountered. Section 5 of this report discusses verification and validation in more detail, especially as they apply to expert systems. An autonomous system architecture must include automated tools to aid the verification and validation tasks.

3.2.2 Functional Specifications

3.2.2.1 ALS Opportunities for Autonomy

In Section 2, candidates for expert system decision applications were discussed under four groups:

- test and checkout diagnostic aids
- on-board health and status monitoring functions
- pre/post launch analysis
- mission planning and operations

In this section an integrated systems engineering approach is used which identifies broad functional areas of opportunity for autonomy which include the previously identified candidates, and which also provides a structure for uncovering other opportunities.

Opportunities for benefits from increased autonomy of operation may be revealed by examining the functions to be performed during the life cycle of the ALS. The functions performed are categorized in Table 3.1 and the phases of the life cycle are listed in Table 3.2. These functions occur in both the vehicle and operations segments of ALS. One must distinguish between these segments and the physical allocation of these function to an on-board system versus the ground. The first Three functional areas are operations related; management, engineering, and engineering support functions; whereas the ALS vehicle functions are, as the name implies, related to the vehicle segment. For an unmanned vehicle, all of the vehicle functions must be automated or else they must be re-allocated to the ground.

Table 3.3 decomposes the categories of project functions. There is a commonality of the sub-categories of functions as shown in this table. That is, monitoring and control functions are allocated to both the ground and the vehicle. The distribution of this allocation to the vehicle versus the ground corresponds to the degree of autonomy of the vehicle relative to the ground.

1.0 Management Functions
2.0 Engineering Functions
3.0 Engineering Support Functions
4.0 ALS Vehicle Functions

Table 3.1—Top Level Project Function Are Organized Under Four Categories

1.0 Development
1.1 System definition
1.2 Demonstration/validation
1.3 Full-scale development
2.0 Production
2.1 Initial launch system production
2.2 System improvements
2.3 Full-scale launch system production/delivery
3.0 Operations
3.1 Pre-launch operations
3.2 Launch operations
3.3 Ascent operations
3.4 Recovery operations
3.5 Post-flight operations
4.0 Disposal
4.1 Salvage
4.2 Recycle
4.3 Junk

Table 3.2—Launch System Life Cycle Divides into Four Phases

The phases of the life cycle, except for disposal, are expanded in Table 3.4 to show the top level tasks in each phase. For example, the task to prepare the vehicle for launch is shown under pre-launch operations. This vehicle related task may be fully autonomous to the vehicle, with only status information sent to apprise the ground of progress, or some of the control could be retained on the ground. An advantage of autonomous operation is that various vehicle configurations may do these operations as required for the specific vehicle. A disadvantage is that the common operations would be duplicated on each vehicle instead of being implemented only once on the ground, which is a more costly hardware acquisition. This illustrates that trade studies are essential to the decision process for allocation of vehicle related functions to either the vehicle or the ground.

The functions to be performed during the life cycle may be viewed as a matrix. The shell of this autonomy opportunity matrix is shown in Table 3.5. There are opportunities to automate/computerize parts of virtually every one of these approximately 400 cells, with each cell possibly having multiple opportunities. Because of limitations on time and other resources, this section will concentrate on one important sub-matrix of opportunities: the functions on the ALS vehicle during the operations phase of the life cycle.

This study focuses on the sub-matrix of the operations phase and the ALS vehicle functions as shown in Table 3.6. The opportunities to provide autonomous operation of the vehicle in this area offer enormous benefits in lowering the cost of operation and reducing the turn around time of the vehicle, our primary goals. It should be noted that

these opportunities for autonomous operation of the vehicle must be planned and implemented in the development phase.

1.0 Management Functions	3.0 Engineering Support Functions
1.1 Plan/Organize	3.1 Operate system
1.2 Schedule tasks	3.2 Monitor system
1.3 Staff project	3.3 Control system
1.4 Monitor progress	3.4 Maintain system (including databases)
1.5 Control project	3.5 Train personnel
2.0 Engineering Functions	4.0 ALS Vehicle Functions
2.1 Analyze/simulate	4.1 Perform operations
2.2 Design	4.2 Monitor operations
2.3 Build	4.3 Control operations
2.4 Test	4.4 Update system databases
2.5 Integrate	4.5 Manage fault tolerance
2.6 Document	4.6 Report operations

Table 3.3—Decomposition of Project Function Reveals Common Sub-Functions

In the focused matrix, there is included the opportunity to do autonomous tests of subsystems of the vehicle while they are on the vehicle. The test may be initiated, for example by either a specific off-board request or as a result of something that occurred during an integrated (autonomous) vehicle test. Test data analysis could be implemented on-board, to be conducted in support of off-board monitoring of the vehicle status. Sensor calibrations even today are routinely done automatically once an off-board command is received.

It is noted that several of these operations are conducted in different sub-phases. Many will also occur in other phases of the life cycle, so that benefits of their autonomy is compounded in further cost/time savings.

Inspection of the focused matrix shows that a primary area of opportunity lies in the performance of pre-launch operations. What the matrix does not show is the time scale; launch operations are paced by the very items that are listed as autonomy opportunities. Any of these items that can be done faster or in parallel as a result of autonomous implementation will shorten the pre-launch phase. This is significant, do to the short stay on pad.

Another vehicle autonomy opportunity is in the screening of telemetry data. This will translate into a reduction in space/ground communications, which is a major cost element in a launch system.

There are other areas of opportunity for autonomous vehicle operation that are not included in this focused area of the matrix. Some of these have been discussed previously in Section 2. Further, there are other opportunities for knowledge-based methods/autonomy of operation associated with the ALS which are not candidates for implementation on the vehicle itself. (On the other hand, not all of the autonomous vehicle opportunities may best be implemented totally or even in part on the vehicle; trade studies need to be conducted to determine the best allocation of those operations which are first determined to be cost beneficial to implement autonomously.) Operations such as pre-launch configuration of the launch vehicle/payload and their interfaces, the updating of procedures based on the outcome of previous operations, and the CAD/CAM procedures in the vehicle/system design

1.0 Development

1.1 System definition

1.1.1 Mission need determination

1.1.2 System concept exploration

1.1.2.1 Perform system concept studies

1.1.2.2 Prepare preliminary specification

1.1.2.3 Define life cycle cost

1.1.2.4 Analyze logistics support

1.1.2.5 Prepare system concept paper

1.2 Demonstration/validation

1.2.1 Perform system requirements trade studies

1.2.2 Prepare system definition

1.2.3 Perform risk analysis

1.2.4 Prepare baseline allocation

1.2.5 Perform risk analysis

1.2.6 Identify/demonstrate required technology

1.2.7 Analyze "ilities" (reliability, producibility, maintainability, supportability)

1.2.8 Prepare validation and verification plans

1.2.9 Prepare decision coordination paper and integrated program summary

1.3 Full-scale development

1.3.1 Perform detailed design/ specifications

1.3.2 Produce baseline product

1.3.3 Conduct test and evaluation

1.3.4 Perform system validation/ effectiveness tests

2.0 Production

2.1 Initial launch system production

2.1.1 Build/test/deliver

2.1.2 Provide logistics support

2.2 System improvements

2.3 Full-scale launch system production/ delivery

2.3.1 Build/test/deliver

2.3.2 Update system

3.0 Operations

3.1 Pre-launch operations

3.1.1 Plan missions (sorties)

3.1.2 Prepare vehicle

3.1.3 Prepare ground system

3.2 Launch operations

3.2.1 Integrate vehicle and ground operations

3.2.2 Monitor/evaluate system readiness

3.2.3 Conduct launch/abort operations

3.3 Ascent operations

3.3.1 Monitor/evaluate system performance

3.3.2 Conduct trajectory prediction/abort operations

3.4 Recovery operations

3.5 Post-flight operations

3.5.1 Evaluate vehicle performance

3.5.2 Status/refurbish recovered systems

3.5.3 Report mission successfulness

Table 3.4—Expanded Outline of the Phases of the Launch System Life Cycle Shows the Top Level Tasks in Each

Functions to Perform:	ALS Life Cycle Phases													
	1.0 Development			2.0 Production			3.0 Operations					4.0 Disposal		
	1.1 System def.	1.2 Dem/val	1.3 FSD	2.1 Initial prod.	2.2 Sys. improv.	2.3 FS prod.	3.1 Pre-launch operations	3.2 Launch ops	3.3 Ascent ops	3.4 Recovery operations	3.5 Post-flt operations	4.1 Salvage	4.2 Recycle	4.3 Junk
1.0 Management functions														
1.1 Plan/organize														
1.2 Schedule tasks														
1.3 Staff project														
1.4 Monitor progress														
1.5 Control project														
2.0 Engineering Functions														
2.1 Analyze/simulate														
2.2 Design														
2.3 Build														
2.4 Test														
2.5 Integrate														
2.6 Document														
3.0 Engineering Supt. Functs.														
3.1 Operate system														
3.2 Monitor system														
3.3 Control system														
3.4 Maintain system														
3.5 Train personnel														
4.0 ALS Vehicle Functions							Zoomed in area							
4.1 Perform operations														
4.2 Monitor operations														
4.3 Control operations														
4.4 Update system databases														
4.5 Manage fault tolerance														
4.6 Report operations														

Table 3.5—Shell of Autonomy Opportunities Matrix

	3.0 Operations Phase of ALS Life Cycle				
4.0 ALS Vehicle Functions	**3.1 Pre-launch ops.**	**3.2 Launch operations**	**3.3 Ascent operations**	**3.4 Recovery operations**	**3.5 Post-flt operations**
4.1 Perform operations	• Vehicle subsystems tests • Integrated vehicle tests • Test data analysis • Sensor calibrations • Vehicle status evaluation • Maintenance support	• Power/pressure application • Integrated vehicle tests • Test data analysis • Propulsion system support • Communications verification • Sensor calibrations • Launch decision support • Range safety support • Vehicle status evaluation	• Screen the T/M data • Range safety support • Vehicle status evaluation	• Recovery support	• Maintenance support • Flight data analysis
4.2 Monitor operations	• Vehicle subsystems tests • Integrated vehicle tests • Sensor calibrations • Test data analysis • Vehicle status	• Power/pressure application • Integrated vehicle tests • Test data analysis • Propulsion system • Communications • Vehicle status	• Vehicle status • Select data for display • Propulsion system • Avionics system	• Vehicle status • Select data for display • Avionics system	
4.3 Control operations	• Vehicle subsystems data • Integrated vehicle data • Sensor calibrations data	• Power/pressure application • Integrated vehicle tests • Test data analysis • Propulsion system • Communications	• Avionics configuration • Propulsion system • Abort/alternate mission	• Avionics configuration • Propulsion system • Abort/alternate mission	
4.4 Update system databases	• Vehicle subsystems tests • Integrated vehicle tests • Sensor calibrations • Test data analysis				• Vehicle subsystems data • Integrated vehicle data
4.5 Manage fault tolerance	• Fault detection/isolation • Fault explanation • Test data analysis	• Fault detection/isolation • Fault explanation	• Fault det/iso/correction • Fault explanation	• Predicted impact	• Fault explanation
4.6 Report operations	• Processing logger • Vehicle subsystems tests • Integrated vehicle tests • Test data analysis • Sensor calibrations • Fault detection/isolation • Vehicle status evaluation	• Processing logger • Integrated vehicle tests • Test data analysis • Fault detection/isolation			• Flight reports • Fault resolution report

Table 3.6—Focused Area of Autonomy Operations Matrix

are candidates for autonomy off of the vehicle. One important area of opportunity for autonomy is in the planning of the mission/flight trajectory. Much of this work has become specialized, and there are experts whose knowledge can be drawn on to incorporate into knowledge-based applications. The capabilities of various autonomous operations would be enhanced by an ability to access the ALS Model (ALSYM) to perform what-if analyses or forecast system status. These areas need to be evaluated in terms of cost versus benefits; the potential for cost savings may be even greater than for the area shown in the focused matrix.

3.2.2.2 Integrated Autonomous Control

A maximally autonomous system would include the operations identified in the opportunities matrix in the previous section. For efficience of data flow and economy of costs there is strong argument for these functions to each operate in a semi-autonomous manner, sharing information as needed. Some of the functions identified may invoke others to perform sub-tasks. Some overall autonomous control function, possibly using expert system methodology, might also be part of a maximally autonomous system.

The need for a general approach to provide autonomous operation of a complex interactive set of functions such as those identified in the previous section is readily apparent. This approach should provide a method by which functions could operate autonomously by probing other areas of the system to keep track of events which affect the operations to be performed. Such an integrated autonomous system could share datathrough direct real time communication and through the ALS Unified Information System (Unis). The R/T network and Unis can be though of as global blackboards on which all shared data is posted. Functions could establish probes or triggers to be automatically notified when certain events are recorded.

The ability to access centralized mathematical models such as the ALS Model (ALSYM) in order to perform what-if analysis to check out autonomous plans can greatly enhance the degree of autonomy that can be achieved. Vehicle monitoring functions could use simulation data as a reference to compare sensor data to. Simulations based on data recorded from previous missions could be used to predict failures before they occur.

3.2.3 Risks

The risks involved in developing a maximum autonomous system can be minimized by selecting only those functions which are determined to be appropriate to automate and by providing the capability for human override of critical functions. Selected portions of current launch processes have already been automated in existing systems. Fault tolerant systems have been built using redundant hardware. The main risks are incurred in allowing the system to make decisions tthat effect the vehicle without human intervention or to automatically take steps to safe itself in an anomilous event. By reserving a capability for emergency override by a human, and with a hierarchial approach towards abstraction and management, the safety risks can be mitigated.

Autonomous interplanetary spacecraft have been in use for a number of years. These spacecraft are able to make critical maneuvers without any intervention from the ground. The experience of these spacecraft reinforces our belief that autonomous control can be incorporated into ALS.

3.3 TRADES

3.3.1 Expert Systems vs. Conventional Programming

Autonomous systems are analogous in some ways to expert systems. An autonomous system can be thought of having a goal which it is trying to achieve. The guidelines for accomplishing the goal can be expressed as the rules of an expert system. The expert system uses its rules to try to prove the goal (i.e. it develops a plan). The expert system checks the configuration, issues control commands, and reads sensor inputs to determine facts. The expert system backtracks and tries an alternate method of proving its goal when an anomaly results in the negation of some rule. Rules must be written with some skill involved when converting guidelines to rules. Mapping the high level definition of autonomy to an expert system may not always be appropriate.

When we take a closer look at autonomy, we see that it involves the functions of planning, controling, monitoring, diagnosing, and configuring. These functions are among those for which expert systems have been proposed. Examples of successful expert systems exist in each of these areas.

Many of the opportunities for autonomy presented in the matrix in Section 3.2.2.1 may be implemented wholly or in part using conventional procedural methods. Some of the operations presented in the matrix have already been automated using conventional programming in existing vehicle processing and launch control systems. For those operations which can be specified as well defined procedures, conventional systems may be more cost effective. With the current technology, some real-time operations may be easier to implement with conventional systems.

One area which stands out as a likely application for knowledge-based methods is vehicle fault tolerance management. Even in procedural applications of redundancy management, there are numerous logical tests based on heuristics. When failures do occur in such procedural-based systems, the explanation for the system reconfiguration requires enormous resources and delays. The inherent fault explanation capability of knowledge-based methods may alone justify the development investment by preserving the KB for life cycle maintenance.

Table 3.7 compares the features of conventional systems and expert systems. For those operations which are highly time critical or which have well defined procedural specifications, conventional programs should be used. For those operations where experts use rules and heuristics which are likely to change as experience is gained with ALS, expert systems will be more cost effective.

A maximum autonomous system would best be implemented using a combination of conventional and expert systems. Autonomy may best be achieved by implementing cooperating expert systems which perform these lower level functions. The integration of conventional and an expert system has been demonstrated by General Dynamics in a real time application environment. The combination allows the ES to prescribe tasks based upon a decision space and the conventional software exicutes the task by interfacing with the external environment.

3.3.2 On-board VMMS vs. Ground

3.3.2.1 Factors

In analyzing the allocation of functions between an on-board VMMS and ground equipment it is necessary to examine several factors. The traditional concerns of weight and electrical power requirements in limiting the amount of on-board computing are no longer as great as they once were. Nowadays, hardware is small and light weight and power requirements are well within limits, so even with an expendable launch vehicle other factors will determine whether functions are allocated on-board or on the ground. In an autonomous system, we must consider the need for redundancy. By having multiple machines on-board, we can recover from failure of one or more machines. If there is a risk of loss of communications, certain autonomous functions will have to be on-board. Real-time considerations may also require allocation to the on-board computers as there is a time penalty in ground to vehicle communications. Specific mission requirements may also effect the allocation of functions between the on-board system and the ground.

3.3.2.2 Philosophy

This document does not specify the maximum degree of autonomy that can be allocated to the on-board VMMS because it depends on identification and analysis of the factors described above. The most cost-effective functional allocation between the on-board VMMS and the ground should be determined by a detailed analysis of required functions followed by trade study of the above factors. The ALS Model (ALSYM) should be used in this trade study. The functional allocation may be different for different vehicles, with the ground system being configured to match a particular vehicle.

The on-board system should include: guidance and navigation, flight control and sequencing, anomaly recovery, flyback control of recoverable modules, vehicle health monitoring, and communications. This maximizes on-board autonomy while minimizing on-board power, weight, and telemetry bandwidth. Vehicle health monitoring may include predicting and compensating for sensor drift. Anomaly recovery will include handling one engine out and may be able to handle faulty attitude control thrusters if they are designed so that others can compensate.

The proposed maximum autonomous architecture must allow flexible allocation of functions between ground and on-board systems.

	EXPERT SYSTEMS	CONVENTIONAL SYSTEMS
Analysis Methodology	Analyst captures domain knowledge in natural language like rules. Domain expert can provide abstract "rules-of-thumb" rather than detail specifications.	Analyst must develop functional specification of system. Misunderstanding between analyst and domain expert often results in problems later on.
Problem Representation	Explicit symbolic model of the problem. Can be understood by the domain expert. Refined iteratively through prototypes.	Implicitly tied up with model of computer. Only understood by programmer. Fixed at design stage.
Performance	Rules guide focus of attention. Search is narrowed by heuristics. Non-deterministic flow of control.	Flow of control is programmed. Each possibility must be checked in turn. Exhaustive search is intractable
Real-Time Suitability	Current shells must develop real-time controls (priority interrupts, multi-tasking).	Many real-time systems exist. Modern multi-processor systems require extensive s/w tailoring
Flexibility	Knowledge base can be reused. Can learn (or be taught) new rules to adapt dynamically.	Data is intimately tied to the program. Reusability must be planned. Ability to change and adapt must also be planned and adds complexity.
Maintainability	Modular chunks of knowledge (e.g. rules) aids independent modification. Expert recognizes problem representation and can help refine rules.	Modularity must be planned. Code is unrecognizable to expert.
Development	Quick feedback on accuracy of rules. Easy to modify knowledge base, a few chunks at a time. Programmer concentrates on problem representation. Explanation facility helps verify requirements (knowledge chunks).	Long time until requirements and design are finalized, after which domain expert is not involved. Changes to requirements often imply reworking of design. Programmer must concentrate on mapping the problem onto machine representation. Debugging statements reflect implementation rather than requirements.
Verification & Validation	Verification checks that the knowledge base is complete and traceable to the requirements. Validation checks that the system meets the requirements.	Verification checks traceability of each step of development to the previous step. Validation checks that the completed system meets the requirements.

Table 3.7—Expert Systems and Conventional Programs

3.3.3 Distributed System Trades

As shown in the previous section, a maximum autonomous ALS would be distributed between an on-board VMMS and associated ground equipment. In addition, both the on-board and ground systems will most likely be distributed among several machines. The distributed components of a maximally autonomous system will need to communicate with each other in order to share data. This communication can take several forms. Data can be recorded in a central

database or blackboard. The components of an autonomous system would periodically examine this blackboard to see if any pertinent data have been posted. A more efficient mechanism allows components to install probes or daemons in the blackboard. These probes monitor the data being posted on the blackboard and send messages to the components when pertinent data have been posted. In other cases, components may need to communicate directly with each other. A maximum autonomous system can be built where each component is aware of the other components with which it communicates. A more flexible approach would allow components to communicate with each other by capabilities and interest. With this approach, each component registers its capabilities and interest in a global finder database. When a component needs to communicate it looks for a component with the needed capability or interest in the database and communicates with that one. It might also choose to broadcast information to all interested parties. Various communications approaches involve trades, usually trading flexibility for efficiency. An architecture which supports the different approaches will allow the designer of a maximally autonomous system to select the best approach for a given situation. The more flexible approaches allow a higher degree of autonomy in each component.

3.3.4 Object Oriented Design

Object oriented programming can provide numerous benefits if used as part of a maximum autonomous system architecture. The object classes developed using an object oriented approach become building blocks from which the maximally autonomous system is built. This approach allows the same code (building blocks) to be reused in different parts of the system (or in different configurations of the system). This can result in reduced development costs. By using a language which supports object oriented programming, static compile-time checks can verify that the correct objects and interfaces are being used. Additionally, special objects can be provided to support dynamic run-time debugging of the system. Object oriented systems are extensible. New object classes can be derived from existing classes when needed to perform new functions. These new object classes inherit interfaces from their parent classes. They can also inherit code if appropriate. Section 3.2.7.6 of the ALS System Requirements Document mandates that "software developed for Unis shall use the object-oriented software engineering methodology."

3.4 PROPOSED AUTONOMOUS SYSTEM ARCHITECTURE

3.4.1 Knowledge Bus Overview

The proposed architecture for a maximally autonomous system is driven by the implications of autonomy and the trades discussed above. The proposed architecture must be able to support both expert systems and conventional programs. These conventional programs and expert systems must be able to communicate with each other in a distributed environment using a standard interface. The proposed architecture should support such a standard distributed communications interface. By providing a capability for programs to communicate with one another anonymously, the architecture can support flexible functional allocation. A standard way of interfacing the Unis database must be provided. By using an object-oriented design and implementation, the proposed architecture provides extensibility. New object classes may be derived from existing classes when the need arises. These new object classes can share appropriate code from existing classes. Implementation in a high-level language along with the object-oriented design can provide static (i.e. compile time) verification of code. Additional static and run-time verification and validation tools can be provided as part of the proposed architecture.

The K-bus, described in more detail in Section 4, provides such an architecture for a maximum autonomous system. The K-bus provides an object-oriented layered architecture for cooperating expert systems and conventional programs. The K-bus provides a standard way of interfacing knowledge bases so that they can be shared by different agents. Tools are provided as part of the K-bus for developing a standard user interface.

3.4.2 Use of the Knowledge Bus Architecture

The architecture for a maximum autonomous ALS based on the K-bus would consists of a number of operations implemented as **agents**. An agent is a semi-autonomous intelligent module which communicates with other agents via the distributed communications capabilities of the K-bus. Each agent has a set of capabilities and interests which are made known to other agents through the global **finder** service. Capabilities indicate what questions an agent can answer and interests indicate what information the agent can be told. An agent can communicate with another agent either by asking a question or by telling information. Agents don't need to know of each other directly, only the capabilities and interests. Agents may also use **probes** to access information about events which occur on the K-bus. Probes provide non-intrusive monitoring of K-bus transactions. Probes monitor the transactions on an K-bus

object and when certain patterns of transactions occur, they can send messages to the agents which installed them. Probes are similar to one rule expert systems. The statements on the left hand side of the rule provide the pattern which defines the events to be monitored. The actions on the right hand side of the rule cause messages to be sent to the agent which installed the probe. Probes can use information bound to variables on the left hand side of the rule in building the messages.

Figure 3.7 shows part of an autonomous system using the K-bus architecture. Autonomous operations, implemented as agents, have installed probes in the Unis database. These probes monitor transactions to the Unis database. When certain patterns of transactions have been made to Unis, the probes will send messages to the agents, which cause a change in the state of the autonomous operations. Agents also communicate with each other by asking and telling. Because this communication is anonymous, using only the capabilities and interests and the global finder service, functions can be allocated to different platforms without impacting other functions. Third party agents use probes to monitor inter-agent communications and well as the interactions between agents and ALS hardware via K-bus **abstract sensors and effectors.**

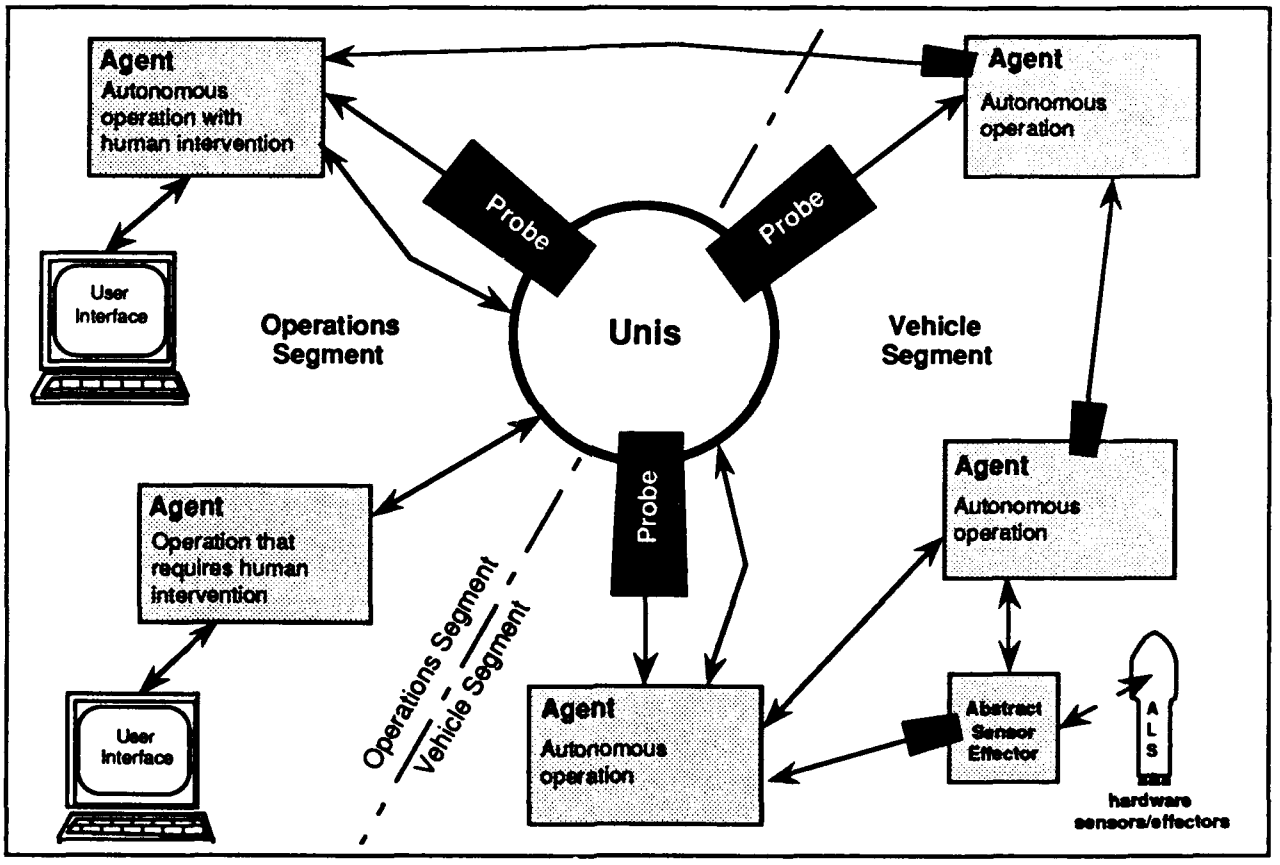


Figure 3.7—The K-bus Supplies An Architecture for Maximum Autonomy Across All ALS Segments

The key to this architecture is the use of Unis as a kind of a blackboard on which all program status is posted. The autonomous agents use probes to monitor those changes in program status which affect their execution. This implies that status information is updated on Unis in real-time. The K-bus supplies a **blackboard** object that can be used to hold time critical status information. A separate agent can be used to update persistent storage in the background, while those agents that need to respond in real-time can probe the blackboard for status changes.

3.5 ASSESSMENT OF PROPOSED AUTONOMOUS SYSTEM ARCHITECTURE

3.5.1 Feasibility and Compatibility

While a maximally autonomous system where after input of the specification for a launch no human intervention is needed may not be feasible, a system can be built in which many operations are automated and where provisions are made for human intervention where this is required. An architecture based on the K-bus provides the framework for achieving a maximally autonomous system. The K-bus should be developed, including the capability to install probes into Unis.

The K-bus architecture will allow new features to be added to the autonomous system over the lifetime of ALS. The K-bus architecture will support both conventional and expert systems which may be needed to implement maximum autonomy.

3.5.2 Benefits

This section describes the benefits of developing a maximally autonomous system based on the K-bus architecture. Among the benefits realized from using this approach is direct or indirect support of many of the requirements stated in the ALS System Requirements Document (SRD). In the following paragraphs, the section of the SRD from which these requirements are derived is shown in parentheses.

3.5.2.1 Integrated System Support

An integrated autonomous system based on the K-bus provides many benefits even though total autonomy cannot be achieved. The proposed architecture ties the components of the autonomous system together via the Unis database. The K-bus provides a standard mechanism for autonomous functions to access the Unis database. Probes can be used to trigger autonomous operations whenever certain events are recorded in the database. Management status and operations control information can be extracted from Unis.

The use of K-bus objects facilitates the management of software development by defining controlled interfaces:

- The K-bus contains a user interface object that can be used to assure the development of a common user interface (3.2.7.4).
- The K-bus distributed database object provides a common interface that assures connectivity between functions (3.2.7.2).
- K-bus objects, such as probes, are useful in developing systems for configuration management of Unis databases and applications (3.2.7.5).

3.5.2.2 Supports Other ALS Requirements

The proposed architecture also supports many other ALS requirements:

- Using simple interface "wrappers", the K-bus supports use of unmodified commercial software (3.2.7.6).
- The object oriented design of the K-bus supports future enhancements without modification to existing software (3.2.7.6).
- Through the use of the distributed database object and probes, the K-bus provides a mechanism to enable Unis capability of storage of all program information (3.2.7.1). The K-bus supports the requirements for Unis data correctness and consistency (3.2.7.3)
- The ALS model (ALSYM) can be viewed as an agent by the K-bus which can be asked to perform analysis by other agents (3.2.8).
- K-bus objects can be developed to support computer system security requirements by enforcing access authorization checking (3.2.6.3).
- The proposed architecture provides improved operating simplicity, and reduces development and operation costs (3.2).

3.5.2.3 Flexible Functional Allocation

The proposed maximum autonomous system architecture allows flexible functional allocation between on-board and ground equipment. The K-bus global name server allows functions to communicate with one another without needing to know the actual location. Flexible functional allocation allows for the support of a variety of on-board system configurations.

3.5.2.4 Built-in V & V Tools

The K-bus contains built-in tools to support verification and validation. These tools include static compile-time checking of interfaces. Other static tools include commercial off-the-shelf software to verify the encoding of knowledge bases. Dynamic run-time verification and validation tools can be built or may exist as part of commercial software that can be used with the K-bus.

3.6 CONCLUSIONS

A maximally autonomous ALS can be developed by automating operations selected on the basis of trade studies of cost and safety criteria. Integrated autonomous operation can be achieved using an architecture based on the K-bus concepts discussed in Section 4. With this architecture, autonomous functions use K-bus distributed communications facilities to monitor the state of the system and act accordingly. The number of operations which require human interface can be significantly reduced.

4. K-BUS

The K-Bus is a software system architecture and framework for the development, integration and verification of distributed real-time systems. Systems implemented under the K-Bus can include both knowledge-based (expert system) and conventional procedural components. The description of the K-Bus includes the following sections:

- Section 4.1 is an overview of the goals, environment, major features and benefits of the K-Bus;
- Section 4.2 is a description of the K-Bus' hierarchical seven-layer architecture;
- Section 4.3 is a preliminary design of the software components that make up the K-Bus;
- Section 4.4 contains trade studies of the issues that have surfaced concerning implementation of the K-Bus, and
- Section 4.5 presents a design and a usage scenario for a hypothetical real-time spacecraft monitoring system based on K-Bus components, for the purpose of assessing the utility of K-Bus functions in actual use.

4.1 OVERVIEW

The K-Bus is a *layered architecture* leading to an *object-oriented implementation* of distributed cooperating systems. The architecture provides guiding principles and structures that will be applicable throughout the lifetime of ALS. The features implemented in the K-Bus provide underlying tools for ease of development and the coordinating functions that permit diverse applications (knowledge-based and procedural) to operate as a coherent system.

The STRESS study of characteristic ALS application areas including situation assessment, process control and inference-directed processing, which are the keys to semi-autonomous space systems, examined many examples of these systems, extracting the common features and analyzing their functional requirements. The K-Bus concept, derived from an earlier *audit probe* concept, provides the required structure and services. The services were enumerated, and the architecture concept for their organization and object-oriented implementation was developed. A literature review of approaches to reusable, reconfigurable distributed knowledge processing systems (see Section 6, Bibliography) supports the concepts of functions and structure.

The remainder of Section 4.1 presents the following: the goals of the K-Bus in terms of ALS mission and life-cycle requirements; the target environment and applications; an overview of the K-Bus architecture; the K-Bus implementation approach; and a summary of the advantages to the program of the K-Bus approach.

4.1.1 Goals of the K-Bus

The K-Bus approach to system development is intended to improve the modularity, reusability, maintainablity and verification and validation of ALS processing systems, which will include distributed real-time knowledge-based or inference-directed systems. The approach should be flexible enough to allow simple replacement of plug-in software and hardware modules, allowing future technology upgrades and the use of off-the-shelf components. These goals derive from projecting ALS program goals onto the development, testing, operation and maintenance of ALS systems.

An optimal ALS system environment should also provide a methodology for writing complex applications that is at a higher-level than exists with current tools. Just as an operating system manages physical resources, the K-Bus should provide means to access common reasoning services for knowledge-based applications and analogous high-level services for procedural applications. For example, expert systems, which currently have to be laboriously hand-crafted even with today's powerful tools, should be assembled from plug-compatible K-Bus components embodying current technology, including inference engines, distributed blackboards and knowledge bases, in the way that inter-compatible subroutine libraries or spreadsheet templates are assembled.

4.1.2 Target Environment and Applications

The K-Bus is targeted at reliable, embedded, real-time distributed systems supporting ALS. Because ALS consists of diverse ground- and space-based components that will evolve over a long lifetime, its support systems – including K-

Bus – have to be engineered to undergo painless technology upgrades in ways that cannot be currently foreseen. The diversity of components will extend from simple sensor-effector monitors to knowledge-based expert control systems, existing procedural programs, test equipment and off-the-shelf products such as DBMS and data analysis packages.

The ALS environment will also require that support systems be relatively autonomous and capable of intelligent decision-making, to reduce the need for a standing army of engineers. The following is a summary of K-Bus requirements that derive from the ALS environment:

- Support embedded applications
- Support real-time performance
- Facilitate technology upgrades during long lifetime
- Permit mixed procedural, knowledge-based and off-the-shelf components
- Support cooperation of autonomous components
- Support control system and sensor-based applications
- Support fault-tolerant approaches
- Facilitate verification and validation

By building these properties into the toolset and underlying architecture, systems utilizing K-Bus components will have the properties (fault-tolerance, cooperation, etc.) without developers having to take special pains at design or implementation time. Figure 4.1 depicts the path from ALS requirements to the K-Bus design concept.

4.1.3 Functional Overview

The K-Bus provides functions for system architects, application analysts, developers, integrators and testers. Subsets of K-Bus services can also be used by other kinds of knowledge-based or conventional systems not specifically related to ALS operations, such as consultation, design and instruction. The following paragraphs discuss the major functions. Figure 4.3 summarizes major K-Bus functions.

4.1.3.1 Distributed System Coordination

A difficult aspect of developing distributed, semi-autonomous cooperating systems is maintaining coordination and control among the components. The K-Bus' *distributed blackboard* function is an implementation of the multilayer blackboard that coordinates many of the most complex distributed expert systems, such as the Distributed Vehicle Monitoring Testbed (DVMT). A blackboard receives and posts messages describing conditions within a distributed system. A blackboard is monitored by probes that interpret the posted data and take action when certain conditions are observed (see next paragraph).

4.1.3.2 Event-Sensitive Monitors

In addition to using blackboards for cooperation, *probes* are triggers or pattern-sensitive monitors that are on the lookout for specific events or patterns of data and pass notifications to intelligent processes that can take suitable action. Probes can be assigned to blackboards (e.g., observing the posting of conditions throughout the system), databases (e.g., observing patterns of data update), communications mailboxes (e.g., observing patterns of message traffic) and expert systems' working memory (e.g., observing the status of a problem being worked on).

4.1.3.3 Knowledge Base Processing

Most knowledge processing tools such as inference engines embody a common core of representations and functions that can be expressed at a high level of abstraction where implementation details are irrelevant. An example is *forward chaining*, a knowledge processing technique carried out by CLIPS, OPS5 and many other inference engines, using the *production rule* (or if-then rule). By identifying these representations and techniques and defining their functionality and interfaces, it is possible to define a library of them which an application can plug into without

perturbation to the rest of the system. Thus, for example, one inference engine can be transparently replaced by another that might have specific performance advantages.

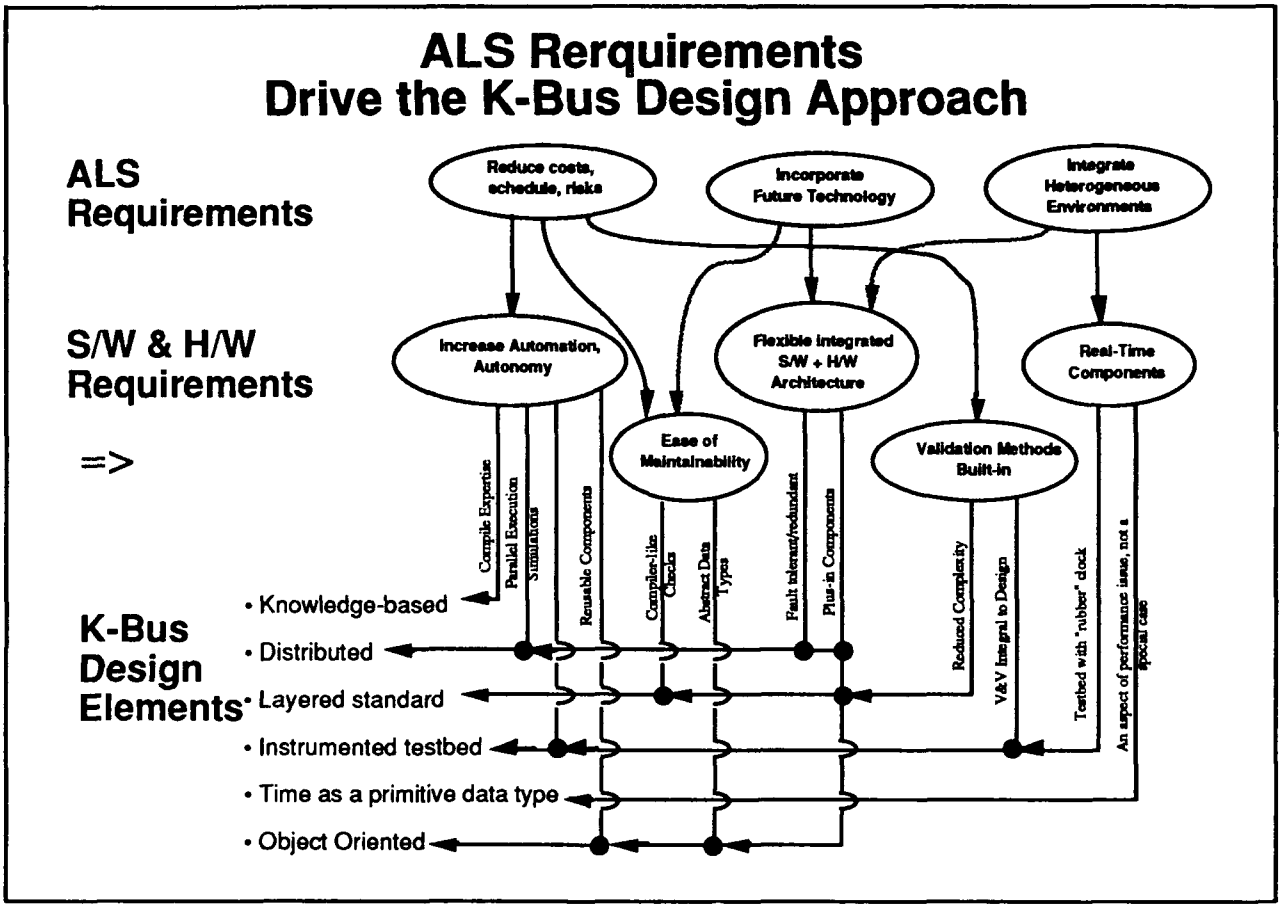


Figure 4.1. ALS Requirements Motivated the K-Bus Design

The knowledge-based processing functions that the K-Bus provides or can provide include the following:

- blackboards with pattern-directed probes (discussed above)
- generic task templates from which prototype applications can be assembled
- temporal reasoning modules for real-time systems
- truth maintenance across multiple knowledge-based components
- inference engines with various strategies such as non-monotonic reasoning and non-standard logics
- transformation functions among various knowledge representations such as rules, frames, scripts and procedures

Viewed from this perspective, the K-Bus is a meta-level knowledge representation language which is portable across applications and across inference engines. Another view is that the K-Bus contains a standardized AI paradigm toolkit, for mix-and-match use by applications on a plug-compatible basis.

4.1.3.4 Intelligent Communications

The K-Bus includes a set of inter-process, inter-machine and inter-network message-handling functions that have an application-level intelligent interface or protocol. This interface provides reconfigurability by hiding details such as the location of the called process. Thus, an application can send messages to "the telemetry front-end" or "the system manager" without knowledge of how they might be implemented or where they might be located within the ALS system network.

The communications functions intelligently work with probes and blackboards to promote coordination. For example, copies of certain types of messages can be automatically sent to blackboards, or an action can be triggered whenever a certain type of message is sent to a certain class of process.

4.1.3.5 Sensors and Effectors

An extension of the intelligent communications function is to provide applications with abstract interfaces to sensors and effectors. The K-Bus provides the means of conveying sensor data to applications and directives to effectors, and a high-level application protocol that shields applications from the low-level hardware-specific nature of these interfaces.

4.1.3.6 System Services

As well as the application-level functions such as intelligent communications, blackboards and probes, the K-Bus also provides the full suite of low-level operating system calls through a protocol that hides the particular OS and machine an application is running on. This supports the K-Bus objective of transparent rehostability from machine to machine, OS to OS.

4.1.3.7 Built-In V&V

The K-Bus accommodates explicit V&V tools, plus other capabilities (e.g., probes) that can be used to advantage in V&V. On the one hand there are static tools such as knowledge base consistency checkers, database analysis tools and compilers that, with the object-oriented implementation, can ensure structural consistency. Compilers can do this through inheritance, where high-level properties are passed down to subclasses of modules without the need for modifications, and through information hiding where included low-level modules can be used with confidence because higher-level code cannot alter their internal workings.

Dynamic tools, on the other hand, perform run-time system monitoring. Probes and verification traps can be set on likely locations like communications interfaces, blackboards and databases to monitor activity and, through pattern-directed inferencing, report on out-of-bounds conditions so that appropriate action can be taken.

4.1.3.8 Testbed Operation

Integral to the K-Bus is a testbed mode which supports examination of both performance (through different implementations of the same services) and reliability (through tools such as static syntactic checkers and dynamic audit probes). The testbed can support multiple parallel version of applications or components that deal with the same data, where one is "live" and the others are "under test," without modification of the components themselves.

The K-Bus can be seen as a CASE tool for (distributed, real-time) knowledge-based systems, in that it supports a development methodology. The methodology is driven by the particular modularization provided by the K-Bus architecture. The architecture points the way for designers in elaborating designs based on the K-Bus. The particular componentization of the K-Bus provides an implementation vocabulary as well as implementation building blocks and interfaces.

It is important to distinguish between the facilities provided by the K-Bus (namely, the protocols, methodology, V & V tools) and the services supported by this architecture. The K-Bus is not a hybrid shell or a distributed AI testbed or an operating system, but a set of tools for using or building such a system.

4.1.4 Architectural Overview

While common features of distributed and real-time systems receive architectural support, what the K-Bus itself provides is the interface to resources which supply these services (inference engines, database managers, etc.): the K-

Bus supplies the cement (and a blueprint) but not the bricks. No commitment to one technique or another is made, although some techniques fit more naturally than others into the K-Bus model of systems organization.

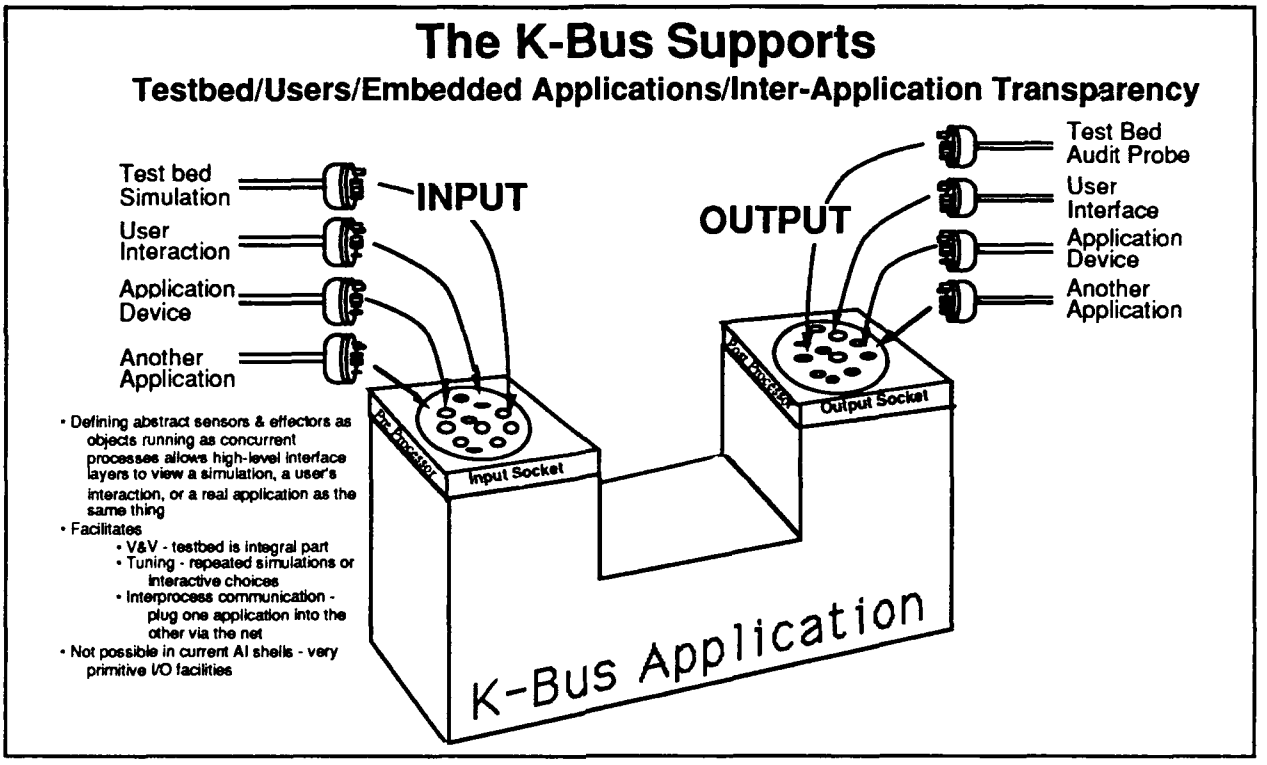


Figure 4.2. Architecture Permits External Agent Transparency

It is important to distinguish between the facilities provided by the K-Bus (namely, the protocols, methodology, V & V tools) and the services supported by this architecture. The K-Bus is not a hybrid shell or a distributed AI testbed or an operating system, but a set of tools for using or building such a system.

The layered architecture is depicted on Figure 4.4.

This concept uses two software engineering paradigms: object-orientation and layering. Viewing system components (such as a reasoning technique) as objects means that its representation and application are encapsulated together, with the implementation hidden from the remainder of the system; this is an extension of the well-known software engineering technique of data abstraction. Communication among objects consists of sending messages according to the client-server approach.

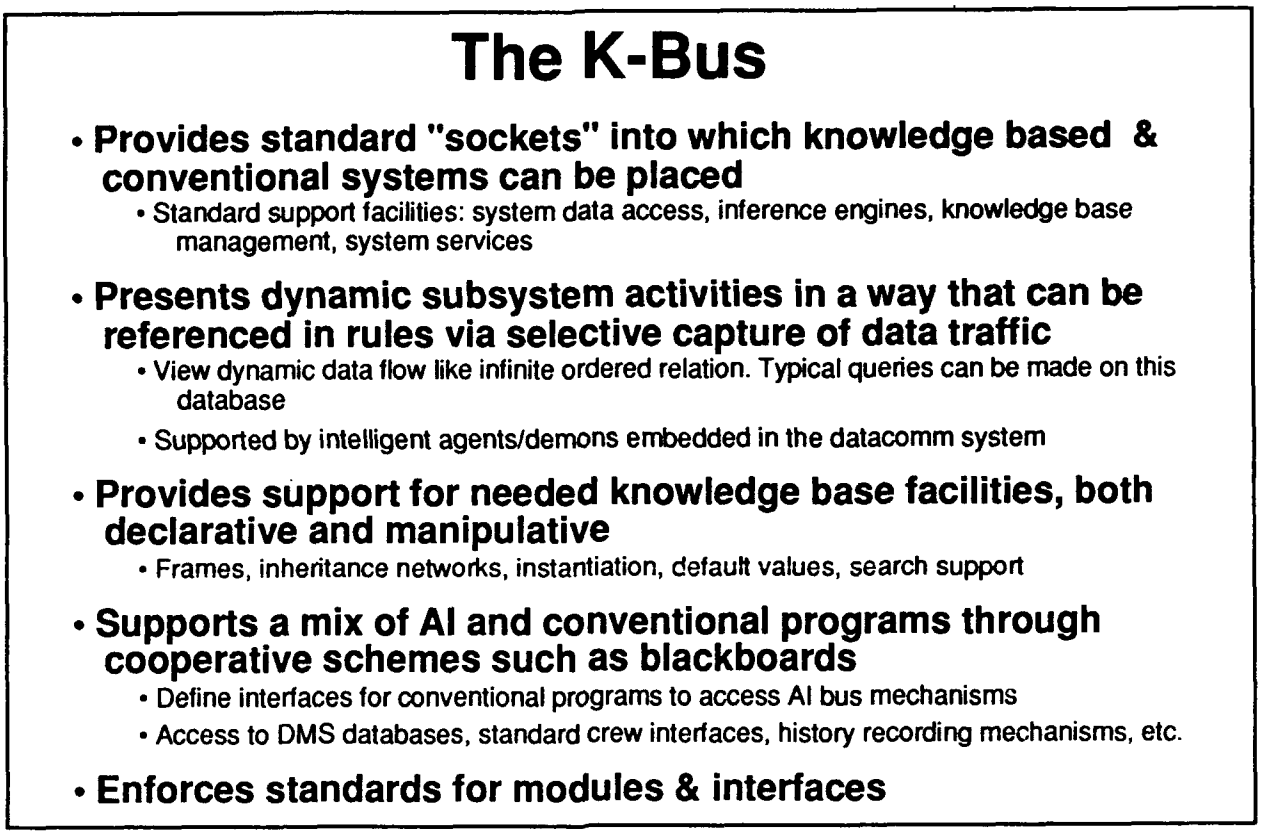


Figure 4.3. Facilities Provided and Supported by the K-Bus

A layered architecture decomposes system functionality required into a sequence of steps, each from well-defined sets of lower level services. At the highest level are systems of cooperating applications (such as *pre-launch checkout*), in intermediate layers such complex services as message routing and sensor interpretation, and in the lowest layers activities close to the hardware and operating system such as reading a serial data line. Each layer provides the appropriate services for the layers above it.

With an object-oriented implementation, each layer of the K-Bus can choose different service libraries for maximum flexibility and modularity. Services in one layer are insulated from changes in the implementation of services they use in lower layers, because the (public) protocols remain the same despite changes in the (private) implementation. Thus, alternative implementations can be selected and software upgrades can be installed without alteration of higher-level modules. Because the choice of libraries is made at compile time, not at run time, there is no performance handicap. Another significant advantage of the object-oriented approach is the ease of writing code as a result of the powerful *inheritance* feature.

For each layer of services, a set of protocols defines the interface to those services. This enforces a structured design while allowing a flexible implementation. Along with the protocols, a set of static verification tools checks the application syntax for possible errors. The separation of the representation language from its implementation permits modular V & V, as does the layered approach. Furthermore, dynamic validation will be supported by the K-Bus audit probes, which are intelligent demons attached to the service objects, to be used in building an instrumented testbed. These audit probes act like stream transducers which monitor and query not only physical transactions (over the network or a database, for example) but also software invocations.

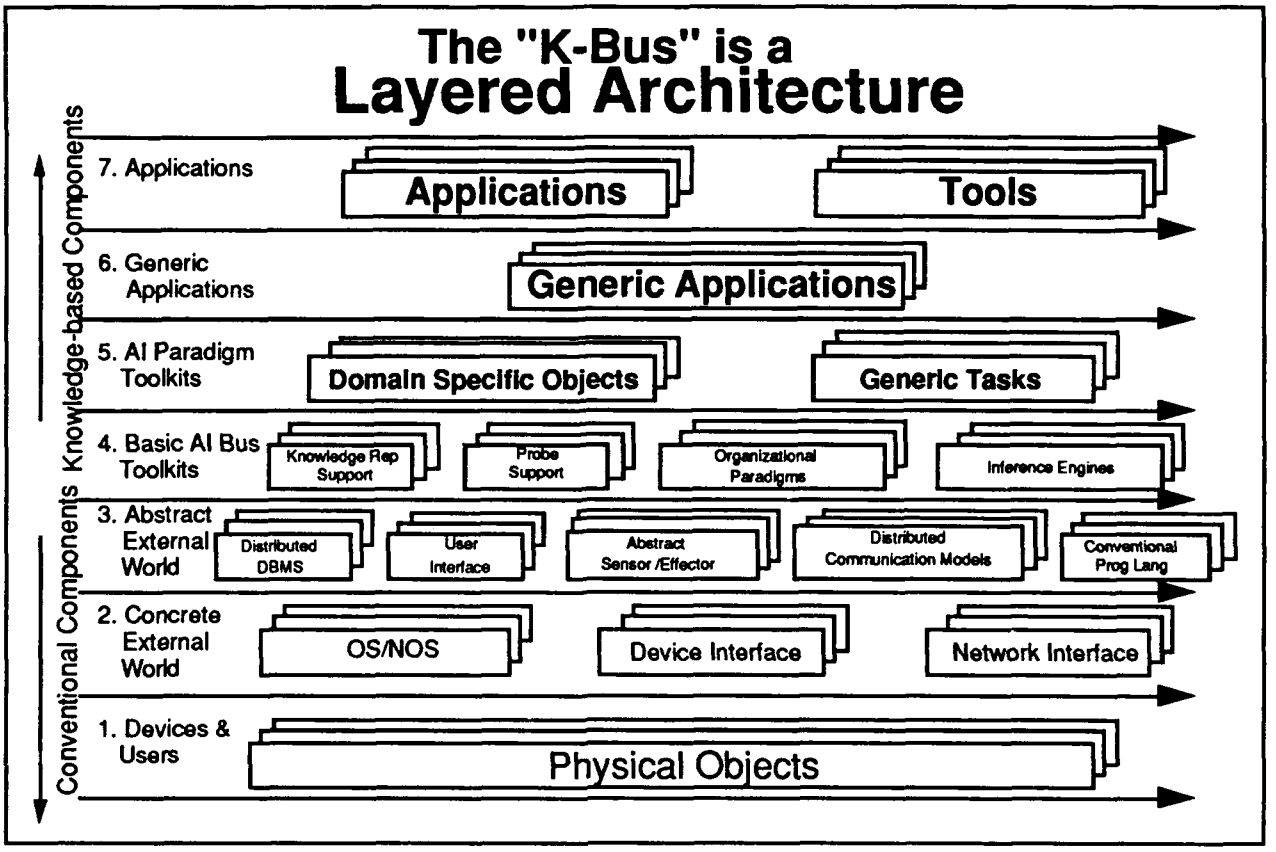


Figure 4.4. The Layers of the K-Bus

Viewing the K-Bus as layers of services reveals it as a CASE tool for (distributed, real-time) knowledge-based systems. It provides specific well-defined software functions and interfaces, and an object-oriented methodology for combining these elements into systems.

4.1.5 Related Work

Other efforts have attacked similar problem areas to that addressed by the K-Bus, particularly work in the distributed artificial intelligence community. Work in the object oriented systems area has had a strong influence on the design approach taken. Part of the K-Bus concept development included a review and analysis of related work. No single system we examined met the multiple goals envisioned for the K-Bus. Many systems examined did have ideas and approaches that we borrowed and integrated into the overall concept. Figure 4.5 lists some of the work that is related to the K-Bus, either in terms of containing technology from which we borrowed, or in terms of attacking similar problem areas. The bibliography of section 6 provides more details concerning related work.

4.1.6 Summary of Advantages

Although there are many ways of looking at the K-Bus, and thus many dimensions to its advantages, the following seem to be among the most significant:

- Life-cycle cost savings
- Technology transparency, ease of upgrades
- Built-in V&V
- Ease of programming, using KBS, mixing KBS with conventional
- Real-time reconfigurability

These advantages are illustrated in the design assessment, Section 4.5, where a usage scenario is discussed in terms of the K-Bus.

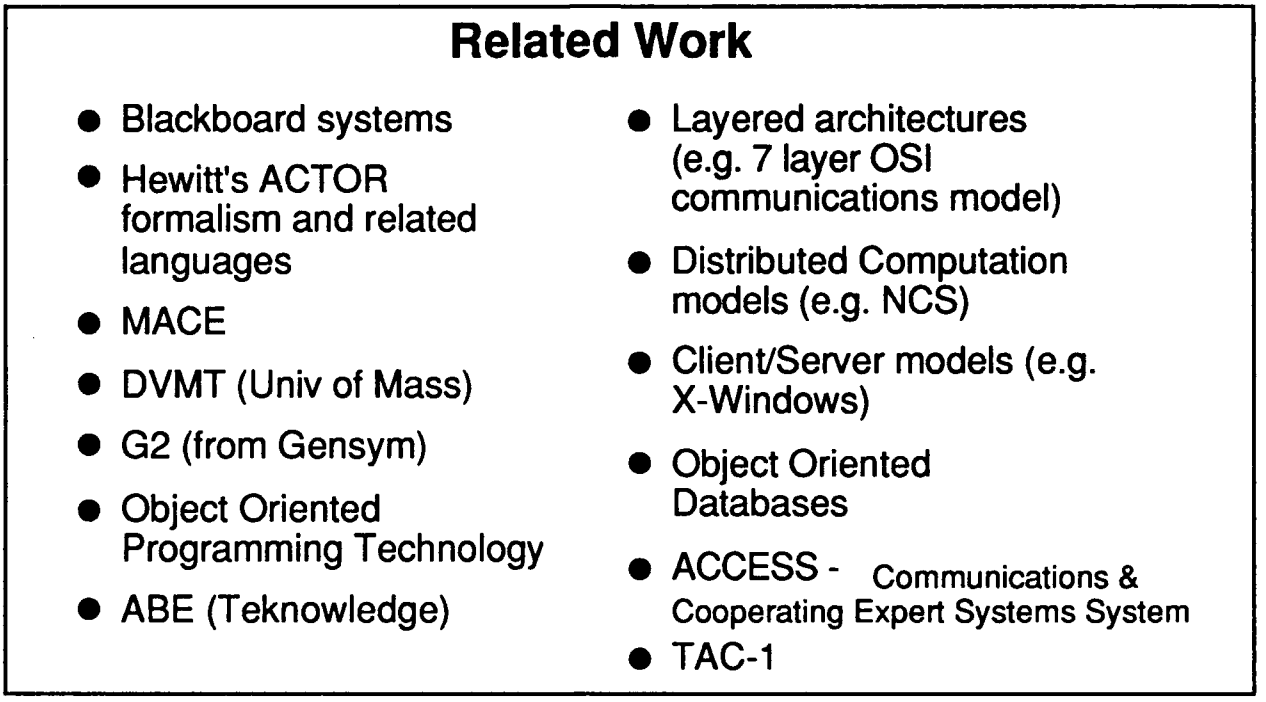


Figure 4.5. The K-Bus Builds on Existing Technology

4.2 LAYER DESCRIPTIONS

4.2.1 Devices & Users Layer

4.2.1.1 Devices & Users Layer Functionality

At this layer lie the physical objects and low level languages (microcode and machine language) which manipulate them. The objects include computer platforms, network physical components, devices being monitored and controlled, devices which do the monitoring and control (physical sensor and effector interface hardware), etc. The user's physical embodiment can be included at this level, in that his inner workings are inscrutable to the rest of the system. If a human's inner cognition is to be considered important (for example, in an intelligent tutoring system) then he would be regarded as a high level agent (application layer).

All these objects will be off the shelf, with no contribution from the K-Bus architecture. An exception, for efficiency reasons, is possible hardware support for probes.

4.2.1.2 Devices & Users Layer Interfaces

This layer presents a low-level, detailed software abstraction of the underlying physical hardware to higher layers. The K-Bus architecture makes only indirect demands, through the Concrete External World layer, with no specific structures or functions except those implied by higher layers.

4.2.2 Concrete External World Layer

4.2.2.1 Concrete External World Layer Functionality

This layer consists of the languages which allow the physical objects to be cleanly modeled as abstract automata. The objects include operating systems, device interfaces, and network interfaces. Most of these objects will be off the shelf, with the possible exceptions of standardized device interfaces, and probe support from network and device interfaces.The extent to which these objects are separate is an issue for the trade study: at one extreme there could be

a single distributed operating system object which gives higher layers a transparent view of the network services. However, this is not currently fully feasible given the state of the technology, and is not necessarily desirable. For example, distributed agents often want to know where other agents are and want to control their own access to system resources. Similarly, an operating system may supply network interface services modeled on its device interface services. For flexibility the three object types in this layer are described separately.

Each of these three types is itself layered, and another issue is to what extent the sublayers are to be represented as separate objects. As this layer's interface with the lower layer is not of interest, the important interface is with the higher layers. This observation means that it is probably sufficient to represent the services by single objects whose internal layering is hidden from clients.

4.2.2.1.1 Operating System Functionality

The operating system handles computer resource allocation and scheduling, and provides file and device access methods and protection. For time-critical applications, a real-time operating system kernel may be desirable, although an agent may prefer to do its own scheduling.

As mentioned above, the operating system may be rich enough to supply the network and device interfaces too. Unix has strong support for device interfaces (in its kernel), but traditionally has been weak in the area of distribution. However, more recently the socket and stream concepts have given local processes a uniform device-like interface to remote processes.

4.2.2.1.2 Device Interface Software Functionality

The device drivers contain all the input/output device-dependent code and thus supply a standard device interface to the operating system. In Unix they are programmed as kernel procedures.

There is one driver per class of devices, grouped together according to hardware idiosyncrasy commonality. One such class may be a probe, which provides network monitoring capabilities in a non-intrusive way.

4.2.2.1.3 Network Interface Software Functionality

The network interface software provides services for communication with remote processors. The OSI 7 layer model is a standard taxonomy for these protocols, but the services themselves are supplied by specific tools (TCP/IP, LU 6.2, etc.). For efficiency reasons, network monitoring via probes may require creating new K-Bus-specific tools: where in the OSI hierarchy these tools are best placed is yet to be determined, based on issues of performance, transparency and ease of installation.

4.2.2.2 Concrete External World Layer Interfaces

This layer presents a virtual machine interface to the higher layers.

4.2.2.2.1 Operating System Interface

The operating system services are enabled from higher layers through system calls. The extent to which each of these calls should be a separate object is an issue of object granularity to be considered in the trade study. The minimal extreme is to have a single operating system object, whose methods supply the system calls based on the syntax of the parameters passed to the object. That approach prevents a proliferation of object types, at the expense of placing all the specification on the syntax of messages.

4.2.2.2.2 Device Interface Software Interface

Device drivers interface with the physical input/output devices (Layer 1) through device controllers, and in turn present a uniform device interface to higher layers. In Unix this device-independent interface is based on the abstraction of the file system, so the major access calls are then Open, Close, Read, Write. The two major device types are block mode and character mode.

4.2.2.2.3 Network Interface Software Interface

The interface with the Layer 1 network hardware is realized at the lowest OSI layer (e.g. RS232). The interface to software objects communicating via the network is designed to hide as much of the physical distribution from those objects as possible. Unix socket and stream facilities present a distributed interface based on its device interface,

which in turn is abstractly modeled on file access. Thus, after initialization, Send and Receive operations for remote processes look like Write and Read for local processes.

The objects in this layer are tools for services in OSI layers 1-4 (e.g. TCP/IP), and Unix sockets. Network interface objects for services above OSI's transport layer can be constructed from these objects, but are not considered stable enough for definition at this time. Higher level distribution schemes, such as Apollo's NCS, fit into the next K-Bus layer where full network transparency is realized. It is at that layer where other policy decisions, such as synchronous vs. asynchronous semantics, are also resolved.

Two kinds of socket objects are needed. The stream socket supplies reliable, sequenced connections whereas the datagram socket is unreliable but may be preferred because of performance benefits; its message sending protocols need an extra parameter for the source address so that calling procedures can perform error-checking. Sockets are installed via Bind and Connect, activated by Listen and Accept, communicated with via Send and Receive, and removed via Shutdown. The message protocols used in Send and Receive are partly specified by parameters to those procedures.

Although the model of inter-process communication is based on message passing, the specification is intentionally vague about the underlying implementation. As a result, parallel architectures receive the same level of support as distributed systems.

4.2.3 Abstract External World Layer

4.2.3.1 Abstract External World Layer Functionality

At this level are the objects which connect the knowledge-based components of an application to operating system resources, databases, the user, and devices being monitored and controlled. A client-server model permits the higher layer client to request a service with no knowledge of how that service is provided by the server, or where the server is physically located (i.e on the local or a remote node). Such network transparency (if desirable) can be achieved through a suitable distributed communication model. Each of the objects in this layer is constructed from objects in this layer, or the adjacent lower layer, using a conventional programming language.

It should be noted that conventional programming languages are included in this layer in Figure 4.4 simply for explanatory purposes. It is not intended that such languages will be represented as objects providing services to other objects, unless it is a standardized way of interfacing with foreign functions which need to be converted to K-Bus format.

4.2.3.1.1 Distributed DBMS Functionality

Database management systems control the secure access and update of persistent information. Commercial products exist that support distributed databases to some extent (e.g. Oracle's SQL*), but true network transparency is not currently possible. The K-Bus supplies the DBMS services in an object-oriented form, by wrapping off the shelf products with an OOP interface, but is not necessarily committed to an OOP implementation of the underlying database system. Such an issue is considered in the trade study. All specified objects can be "checkpointed" to permanent storage, so conversion routines are needed which flatten objects to the DBMS form, and vice-versa.

Databases are one of the objects to which audit probes can be attached, so this capability is added by the K-Bus. Conventional DBMS's often have triggers, actions performed under certain specified transactions, which would give a tightly-coupled implementation of the invocation part of audit probes. The DBMS will also be of use as a primitive blackboard system's data area, i.e. will provide services to Layer 4 organizational paradigms.

4.2.3.1.2 User Interface Functionality

The user interface follows the client-server style exemplified by the X Window system: the application program (client) requests services from the display, without either needing to know the other's physical location in the network. The server allows display access by several clients, typically allocating a window to each (or to each process). The display is bit-mapped and event-driven (e.g. by mouse clicks), the events are queued and processed by window managers.

The user interface objects are themselves layered, ranging from primitive display manipulation, through support for geometric objects (lines, circles, etc.) up to interactive graphic objects such as windows, menus, icons, gauges, etc.

Xlib is a set of low-level routines, examples of the higher level tools are Stanford's InterViews or NASA's TAE Plus system, which supplies a library of interactive graphic objects and the means for creating similar new objects. This layering of tools permits portability across platforms.

4.2.3.1.3 Abstract Sensors/Effectors Functionality

These objects enable hierarchical process control by migrating some of the processing away from the problem-solving level and closer to the device level. They extend the signal-to-symbol transformation of device drivers by providing pre/post-processing servers to include statistical smoothing and scaling, history keeping and pattern matching. History keeping (and calculation of rates and trends) is achieved by maintaining an internal queue of time-tagged data, whereas pattern matching calls for a declarative programming language. Pattern matching is used for event recognition and notification; how much of that capability is provided by the abstract sensors or by attached audit probes is an issue for the trade study.

A client-server style, similar to the user interface, can be used to provide location independence and sharing of sensor/effector servers between several clients. The physical independence enabled by these objects allows transparent substitution of a simulation environment for testing and V & V purposes. Several sensors can be combined to form one abstract sensor; several sensor/effector types may be identified and provided in class libraries.

4.2.3.1.4 Distributed Communication Models Functionality

Different application requirements require different methods for communication between modules, for example message passing vs. shared memory, asynchronous vs. synchronous. The K-Bus supplies tools which support the building of systems structured along standard distribution models. These tools permit building different kinds of "distributed operating systems' from Layer 2 network operating systems, i.e. different ways to achieve network transparency. Layer 2 network interface software was considered to correspond to OSI layers 1-4, a layer 3 communication model provides for OSI application layer tools.

In a distributed environment, objects need unique global names to be used as a reference by other objects. These names will be generated by a single name server object, which itself may be physically distributed; this is similar to a location broker in NCS. Objects communicate with remote objects through local objects called message managers, similar to stubs in NCS. Message managers on the receiving end convert messages to direct calls to the objects, and return results through a similar mechanism.

4.2.3.2 Abstract External World Layer Interfaces

This layer extends the virtual machine view of the lower layer by presenting a logical view of the external world to higher layers. Thus the physical nature of permanent storage, input/output devices and inter-machine connections are hidden from the problem-solving tools.

4.2.3.2.1 Distributed DBMS Interface

The DBMS presents clients with a logical view of the persistent data, to the extent possible with current technology. The interface to the database management system is an object-oriented version of the programming interface, which is well-standardized with SQL. As with operating system calls, discussed above, a trade-off issue is the extent to which a separate object is desirable for each DBMS query. Once again, the minimal number of objects is achieved by specifying the query semantics as methods contained in the DBMS object. In addition to object-flattening routines necessary at this layer, more complex conversion routines are provided at Layer 4 for mapping knowledge-based structures to permanent storage.

Queries are of two types: one which does not return a value, other than an error-code (e.g. delete), and the other which can return a value (e.g. a relational tuple). In general, sets of values can be returned; these can be stored in one data structure, or the query can be considered as a piped query which returns a value and a pointer to the next value.

Queries are considered transactions which may not be atomic, and thus may be aborted by a client. In that case, the actions performed since the transaction was attached are rolled back: in the case of a piped query this may mean rolling back several individual queries. This transaction management, necessary for multiple access synchronization, is supplied by the (distributed) DBMS, and is monitored by probes.

4.2.3.2.2 User Interface Interface

The user interface presents clients with a logical view of the display(s), independent from their physical representation or location. It is itself layered, with X at the bottom and perhaps InterViews or TAE on top. The interface with the adjacent lower layer is through the device drivers and XLib's network services.

The clients interface with the display through structured objects, displaying them via library calls (with parameters for the objects' attributes) and setting conditions for event activation according to the objects' protocols.

The user himself is considered to be a top-layer agent whose interface with the computer program is modeled on the interface between programs. Thus the user interface is analogous to the abstract sensor/effector interface, being event-driven rather than consultative as in traditional AI systems.

4.2.3.2.3 Abstract Sensors/Effectors Interface

The interface with the lower layers is through device drivers, while the upper layers' interface is through the sensor/effector declarative programming language. This language contains primitive sensor/effector objects which can be chosen and combined in a "block-diagram" style, matching types and parameters. For alarm capability, a sensor may be connected directly to an effector, short-circuiting the problem solving modules.

The sensor/effector object types are hierarchical, including classes for analog/digital, unary/nary and whether the client requests data on demand, in a sampled mode, or when externally generated (synchronous or asynchronous).

In Unix, these objects' driver interface can be realized as virtual devices, implemented at the kernel level by stream stacks. An object's parameters include the data source/sink (e.g. driver), scaling requirements, sampling times, limits, initialization and alignment requirements. The internal history queue may be prioritized. Resulting filtered events are delivered to clients according to the specified message protocol and paradigm (interrupt, RPC, etc.).

4.2.3.2.4 Distributed Communication Models Interface

These objects interface with lower layers through network interface software objects, and with higher layers through access to the name server, transporter room and remote method call objects. It may be possible to construct and supply standard communication models, such as NCS, which hide the interface to these objects from higher layers. At a minimum, the higher, problem-solving layers should be unaware as to whether the objects being communicated with are local or remote, unless network latency times, etc. are part of the problem solving definition.

4.2.4 Basic K-Bus Toolkit Layer

4.2.4.1 Basic K-Bus Toolkit Layer Functionality

The Basic K-Bus Toolkit layer contains commonly used software tools for knowledge-based, symbolic reasoning, higher level frameworks (such as blackboards) for integrating reasoning modules, and probes for dynamically auditing transactions. Many representation schemes are possible, such as rules, frames, semantic nets, scripts etc., with corresponding inference engines for manipulating these representations.

4.2.4.1.1 Knowledge Representation Support Functionality

Objects which represent knowledge are manipulated by an inference engine to solve problems specified declaratively. The specifications are executed by symbolic pattern matching between state descriptions, events and actions. Events and state descriptions are inter-transformable, and are represented by statement objects, which may be partially instantiated by wildcards. A rule object relates events and actions, and consists of a priority, trigger and a sequence of actions and consequences. A trigger contains context information, an action causes side-effects (i.e. outside the scope of the inferencing control thread), whereas a consequence changes the inference engine's work space used in making inferences.

In *rule-based* systems, statements correspond to facts and patterns, and have a simple, flat structure. In *logic-based* systems, they are also called facts, but may have an arbitrarily complex structure. In *model-based* systems, statements corresponds to complex objects such as frames and scripts, which may also be used to implement the rule objects. However, that is an implementation decision rather than a conceptual design issue – a statement does not imply consequences, and only invokes database slot-filler actions such as inheriting an ancestor's data. More general

actions, such as querying the user or reading a sensor, may be realized outside of rules by attaching probes to statements; in frame-based languages these procedural attachments are called active values.

Various combinations of trigger and action types yield all the common rule types, such as: Inference, Production, Hypothesis, Bayesian, Constraint, Belief. These types can be defined as subclasses of the rule class; another hierarchy results in particular off-the-shelf languages' statements and rules at the leaves, such as Clips, Ops83, Prolog, etc., with the paradigm classes (logic, frame, rule) in the middle.

The basic module of homogeneous intelligence is a knowledge source object, consisting of a knowledge base (statements and rules), an inference engine, a priority and an active/inactive state flag. Several knowledge sources are combined into a heterogeneous problem solver using an organizational paradigm. Assertions are made by the inference engine, invoked from a message manager - the knowledge source does not communicate directly outside of itself.

4.2.4.1.2 Organizational Paradigm Functionality

An organizational paradigm provides a way to combine different knowledge sources into a cohesive whole. For example, a forward chaining rule-based system may need to interact with a backward-chaining system according to some higher-level problem-solving strategy. Alternatively, a planning module may need to interact with a diagnosis module: this is an example of modular decomposition based on functional differences rather than on knowledge representation (or inference engine control) differences.

A message manager object translates messages to a knowledge source, blackboard or agent into a method call such as assert, and conversely is invoked by such objects to send a message somewhere. The mapping of objects in one knowledge representation class to objects in another (needed for communication in heterogeneous organizations) is achieved simply by coercion operators defined in the objects themselves. Communication between knowledge sources is via statements, in this context considered to be hypotheses and goals.

A blackboard object is one type of organizational paradigm, based on the concept of a global, structured communication space through which knowledge sources communicate anonymously. The structure is a partitioned hierarchy of spaces, corresponding to levels of abstraction of communicated objects. Of particular interest is a transactional blackboard, modeled on distributed database concepts of using atomic transactions to avoids deadlock. A scheduler maintains an agenda of activated knowledge sources, resolves conflicts and runs the one with the highest priority.

Other organizational paradigms includes the dataflow, based on asynchronous message passing, and the procedure, based on synchronous message passing (i.e., procedural invocation). Higher level paradigms such as contract net may also be useful.

4.2.4.1.3 Inference Engines Functionality

An inference engine interprets a collection of knowledge representation objects. It has a workspace consisting of its copy of the knowledge base queried through a pattern matcher, which is also used to decide if a rule's trigger matches the current state. The workspace is updated by consequences of rules as a result of inferencing and response to external events. The inference engine's control strategy determines such things as the order in which rules are considered, and whether rules are interpreted in a forward or backward direction. A scheduler maintains an agenda of activated rules and decides which one to fire based on conflict resolution methods such as first-found, most recent, most complex trigger, highest priority trigger. The truth maintenance component of the inference engine maintains the knowledge base's dependencies and is used for non-monotonic reasoning and explanation.

4.2.4.1.4 Probe Support Functionality

Probes are the basis for access oriented programming: a probe provides non-intrusive monitoring of K-Bus transactions (special kinds of events), and as a result may invoke some actions. A probe is similar to a one rule knowledge source, with a set of statements providing a history and the trigger of the rule providing its activation condition (which may reference the history). The difference is that the inference engine is not part of the probe, but is part of the object to which the probe is attached, for example a database management system. Its essential characteristic is efficiency, to support minimal perturbation to the objects being monitored, however its OOP

definition is not clean. Instead, its definition as a subclass of knowledge source comes from specifying a set of constraints, as follows:

- What objects can a probe be attached to (e.g. database, data communication, knowledge base, blackboard, ...) – these are defined as a class
- What transactions can a probe monitor
- What actions can a probe execute

Examples of efficiency considerations are having an event reference as its trigger (so the inference engine is a simple lookup), and direct support from lower layer constructs. Probes are conceptually considered to execute concurrently with the object being monitored, but that is not a hard characteristic.

4.2.4.2 Basic K-Bus Toolkit Layer Interfaces

The tools at this layer are used at the problem-solving level, with all physical references resolved at the lower layers. Thus the mapping of agents to machines, processes, etc. is considered to occur at Layer 3 and below: no policy decisions of a physical nature are implied by the Layer 4 tools. For example, an agent may be realized as a single process, multiple processes, or share a process with other agents. The Basic AI tools may be combined to form higher level problem-solving abstractions at Layers 5 and 6, or used directly in an application, agent or tool at Layer 7.

4.2.4.2.1 Knowledge Representation Support Interface

The statements and rules in a knowledge source must be compatible, and compatible also with the inference engine – this is the idea of a module of homogeneous intelligence. Knowledge units have coercion operators which enable translation from one representation to another. The interface between knowledge sources, and with the external world (via layer 3 objects) is achieved by the organizational paradigm objects. Statements are considered to be the layer 4 equivalent of layer 3 database records, whether relational (as in Clips) or object-oriented (as in frames). The mapping between databases and statements is also performed by coercion. Statements are interpreted and unified by inference engine objects. Knowledge sources can be initialized, run (for a number of cycles, for a time interval, or until a condition is satisfied), suspended and resumed, and learn new information via the assert method.

4.2.4.2.2 Organizational Paradigm Interface

A message manager uses layer 3 distributed communication objects in order to resolve location and naming references. The message manager has a send method to transmit messages to other objects. It may also need performance parameters (e.g., network latency times) to use in scheduling knowledge sources. The references to knowledge sources are by name. In the case of a blackboard, update and query accesses are similar to layer 3 database protocols, except based on pattern matching supplied by inference engine objects. As well as content modification commands (Add and Delete), blackboards can be dynamically restructured by structure modification commands.

4.2.4.2.3 Inference Engines Interface

A client can request an inference engine to load a knowledge base (i.e., construct its workspace), initialize itself (i.e. construct its agenda), run (for a number of cycles, for a time interval, or until a condition is satisfied) explain a statement, save its work space as a persistent knowledge base, be suspended and resumed, and assert a new statement. Only one knowledge base can be stored in the workspace at a given time: the state of the workspace before the first load or after a second load is an implementation decision. Similarly, the sequence of requests should be load, initialize and run; a different order is incorrect but may default to some action.

It may be possible, and desirable, to decompose the functionality of an inference engine into smaller reusable components. This is an issue discussed in the trade study.

4.2.4.2.4 Probe Support Interface

A probe is executed by the Fire method, called from the probed object when the trigger is matched. Probes are manipulated via the methods attachProbe, detachProbe, enableProbe, disableProbe – these methods are part of the probed object. The physical realization of a probe (for example as a concurrent process) is realized at layer 3, together with support from other lower layer objects optimized for performance.

4.2.5 AI Paradigm Toolkit Layer

This layer contains reusable tools which can be used to write an application in terms of problem solving paradigms identified by researchers in knowledge representation. For example, diagnosis consists of applying several generic task mechanisms such as hierarchical classification, hypothesis matching and abductive assembly. Domain specific objects are the dual of generic tasks, describing the application domain in a form usable by the task languages. These paradigms represent a level of abstraction more powerful than currently available with layer 4 constructs.

An alternative to that top-down view is a bottom-up analysis that calls for the concept of an agent, considered to be a semi-autonomous intelligent module composed from layer 4 objects. The agent is a specialist which communicates externally with other agents via high-level hypotheses. An agent's intelligence can be thus described in terms of layer 4 objects, or in terms of new layer 5 tools coming from research in knowledge representation. These two views can be seen as alternative design philosophies to be resolved by the application developers.

4.2.5.1 AI Paradigm Toolkit Layer Functionality

An agent has a set of capabilities, interests and acquaintances which are maintained by its message manager, and contains component knowledge sources, probes and its organization. It advertises its capabilities (questions it can be asked) and interests (information it can be told), and forms a model of other agents, its acquaintances, by looking at their capabilities and interests.

These objects may also be combined to form more abstract generic task languages. If the generic task approach is taken, domain specific objects encapsulate the parameters passed to the tasks to describe the application.

4.2.5.2 AI Paradigm Toolkit Layer Interfaces

An agent uses two methods for inter-agent communication, Ask and Tell, where Ask returns an answer and Tell just communicates the information. Both are invoked from a member of the agent's acquaintance list, and mapped onto the appropriate component method call (e.g. a KnowledgeSource's Run) through its lists of capabilities and interests. The methods Install and Remove initiate and terminate an agent, respectively.

Inter-agent communication can be demand-driven (through ask), or event-driven (through tell). There is no direct manipulation of another agent's actions (i.e., control-driven) as they can't see each other's internals: all they can do is ask for information, not for actions to be performed.

4.2.6 Generic Application Layer

4.2.6.1 Generic Application Layer Functionality

A generic application uses the domain and task representation from the lower layer, but represents strategies that are reusable in particular problem instances. For example, monitoring the operations of an industrial plant is a generic application which may be instantiated at the highest layer as an application for monitoring a nuclear power plant. Furthermore, a specific application may be a combination of several generic applications. How the generic application is implemented is hidden from the application instantiating it.

4.2.6.2 Generic Application Layer Interfaces

The interface will be the methods Start and Stop, together with a method to initialize and instantiate the particular generic application in terms of task and object combinations.

4.2.7 Application Layer

4.2.7.1 Application Layer Functionality

The application layer describes the task to be done at the highest level of abstraction. A description of the application, consisting of a representation of the reasoning needed and a model of the physical environment (devices and user interface etc), is achieved by using the lower layers of representation. High level languages specify the application in terms of the interaction between application components. At the highest level, the overall VMMS is a combination of multiple conventional and knowledge-based applications, which know how to communicate which each other by using the protocols defined at this layer, which in turn are defined in terms of the lower layer protocols as in the OSI communication model (though at a much higher level of abstraction).

4.2.7.2 Application Layer Interfaces

The methods Start and Stop are used to initiate and terminate an application , respectively.

4.2.8 Examples of Layer Functions

Figure 4.6 shows examples of existing systems which are similar to particular services envisioned for the K-Bus. Many of these could be used directly on the K-Bus to deliver these services, with the proper interfacing encapsulation matching the K-Bus specification.

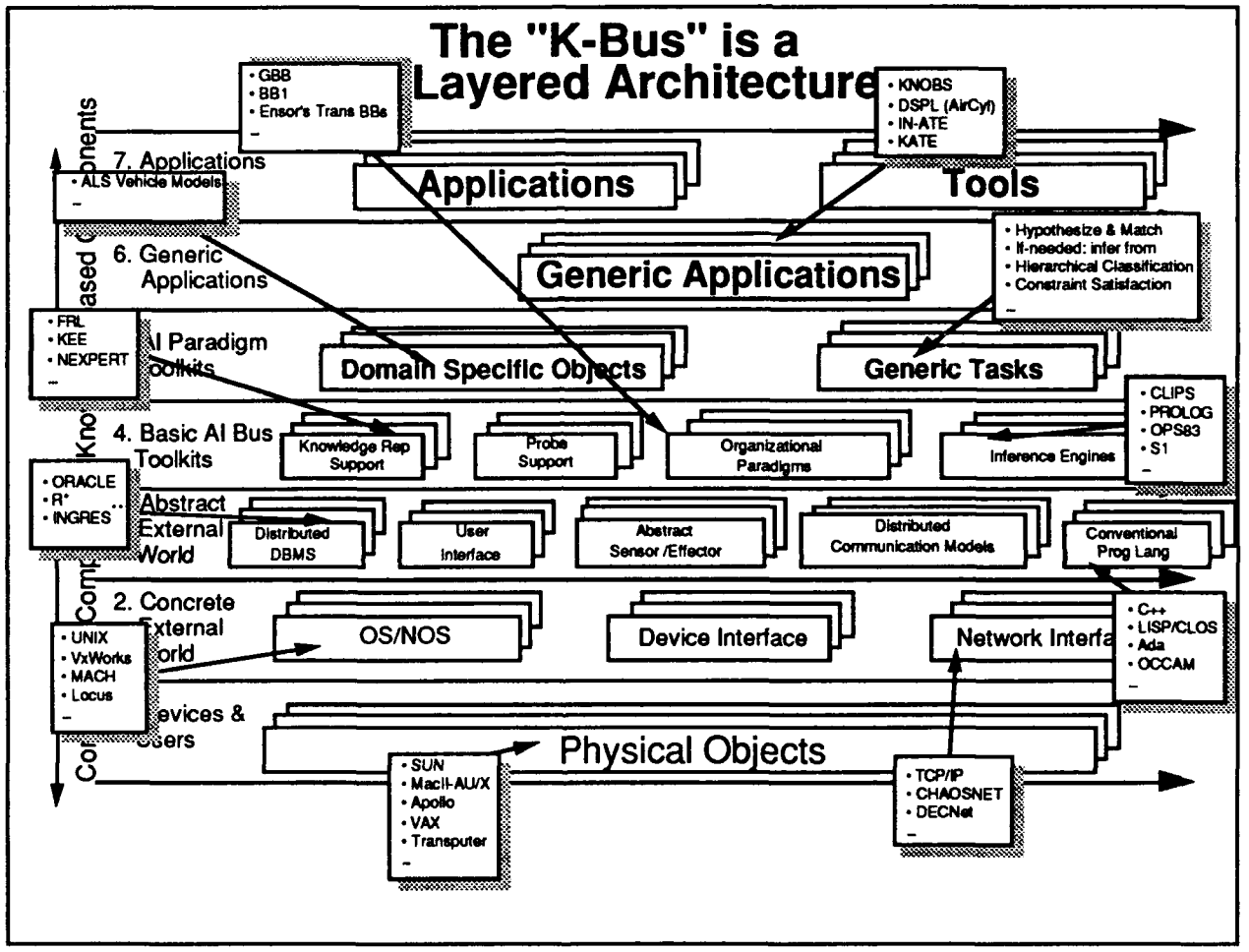


Figure 4.6. The K-Bus Provides a Generic Tool and Component Interface Supporting Many Existing Tools and Components

4.3 K-BUS PRELIMINARY DESIGN

4.3.1 Preliminary Design Overview

The challenge of producing a design for the K-Bus involves making decisions concerning:

- the particular partitioning of the potential solution space into problem solving units, the building blocks to be used by an application designer to construct a system,
- the granularity of the building block, the goal being to make as high level a building block as possible while retaining reusability,

- the interfaces between the building blocks, and the creation of generic reusable interfaces that will adapt to future technology,
- a specification approach that provides maximum flexibility in implementation, yet guides the implementation towards the goals of the K-Bus
- approaches to making use of existing software within the architecture

The preliminary design of the K-Bus concept is specified in the following sections as an object library. The sum of the object classes is the K-Bus Framework. The classes correspond loosely to the layered categories of services identified in Section 4.2.1. A few generic classes, which are never instantiated but serve as superclasses for several classes, span all layers and are specified at the beginning of this section. The object classes specified in the following sections can be seen as corresponding to specific layers of the K-Bus architecture, and to specific functional entities identified in the overview sections. This is not absolute, as the class hierarchy does not directly correspond to the implementation hierarchy.

The objects described provide both an implied implementation architecture, and a standard interface specification. The discussion included with the object descriptions hints at the implied implementation architecture, and a later section makes it more explicit by presenting an application example. Unlike a jigsaw puzzle, where the pieces dictate one particular assembly architecture, the building block objects of the K-Bus must support multiple instances of architectures, being more like an erector set. An hardware analogy that comes close to the software framework provided by the K-Bus is that of a kit composed of a computer backplane, some standard function cards such as memory and processors, device interfaces, etc. and user customizable cards which contain the logic necessary to interface to the bus, and accept standard daughter cards, wire wrap, etc. to allow the application designer to construct an application specific function card. Application systems are then built by selecting the proper combination of the highest level cards that meet the application design requirements, and integrating them in a way to solve the application problem.

It is expected that portable applications would use layer 3 and above as their interfaces. Obviously, the higher level of abstraction that an application can use, the more portable and less prone to needing rework if underlying implementations are changed. A strict rule to good usage is to not implement objects of a lower layer using objects of a higher layer as a building block.

Section 4.5 provides an illustration of how the objects defined here are used to construct a working application suite. The objects specify highly reusable building blocks. The power is in the combinations of these building blocks to realize possible application architectures, all of which will inherit the benefits of the K-Bus architecture: high maintainability, interoperability, and reusability.

The object oriented approach taken provides the packaging for the reusable building blocks mentioned earlier in the report. The object building blocks are highly customizable without changing the object code itself, but instead overriding or appending to its functionality. For the K-Bus, most of the interfaces between components is provided as a standard part of the objects.

Object Library Descriptions

Object Oriented Design bases the system architecture on the objects that the system manipulates and interfaces as opposed to the functions the system elements are to provide. Many of these objects are similar in some respect. In order to reduce redundancy and improve reusability, objects are viewed as belonging to classes, where objects in the same class share some common element or capability. These classes may also share some commonality, and belong to superclasses, resulting in a class taxonomy similar to the biological taxonomy of kingdom, phylum, class, order, etc. Subclasses have all the properties of their superclasses, and add additional properties. Object oriented design also allows subclasses to narrow the properties from those of their superclass. This could be by exception, for instance a class birds, and a subclass of birds-that-can't-fly.

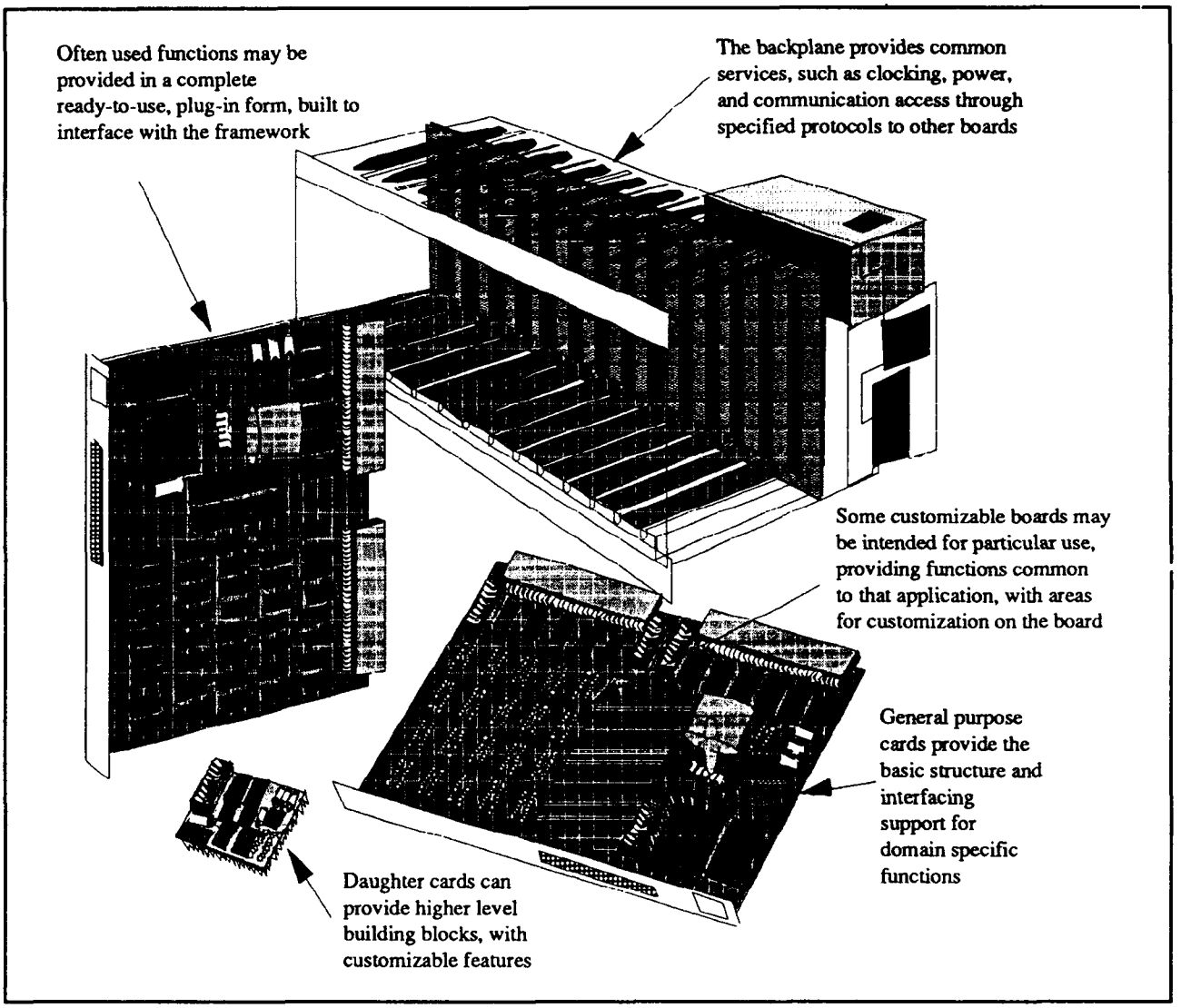


Figure 4.7. Although Software, the K-Bus is in Many Ways Analogous to a Hardware Backplane Bus Standard, Card Cages, General Purpose Boards, and Sub-components

The class taxonomy for the K-Bus is shown in figure 4.8. The hierarchy shown is not a pure hierarchy, in that objects may have multiple superclasses. This makes it possible to share components and functionality with less repetition than a pure hierarchy.

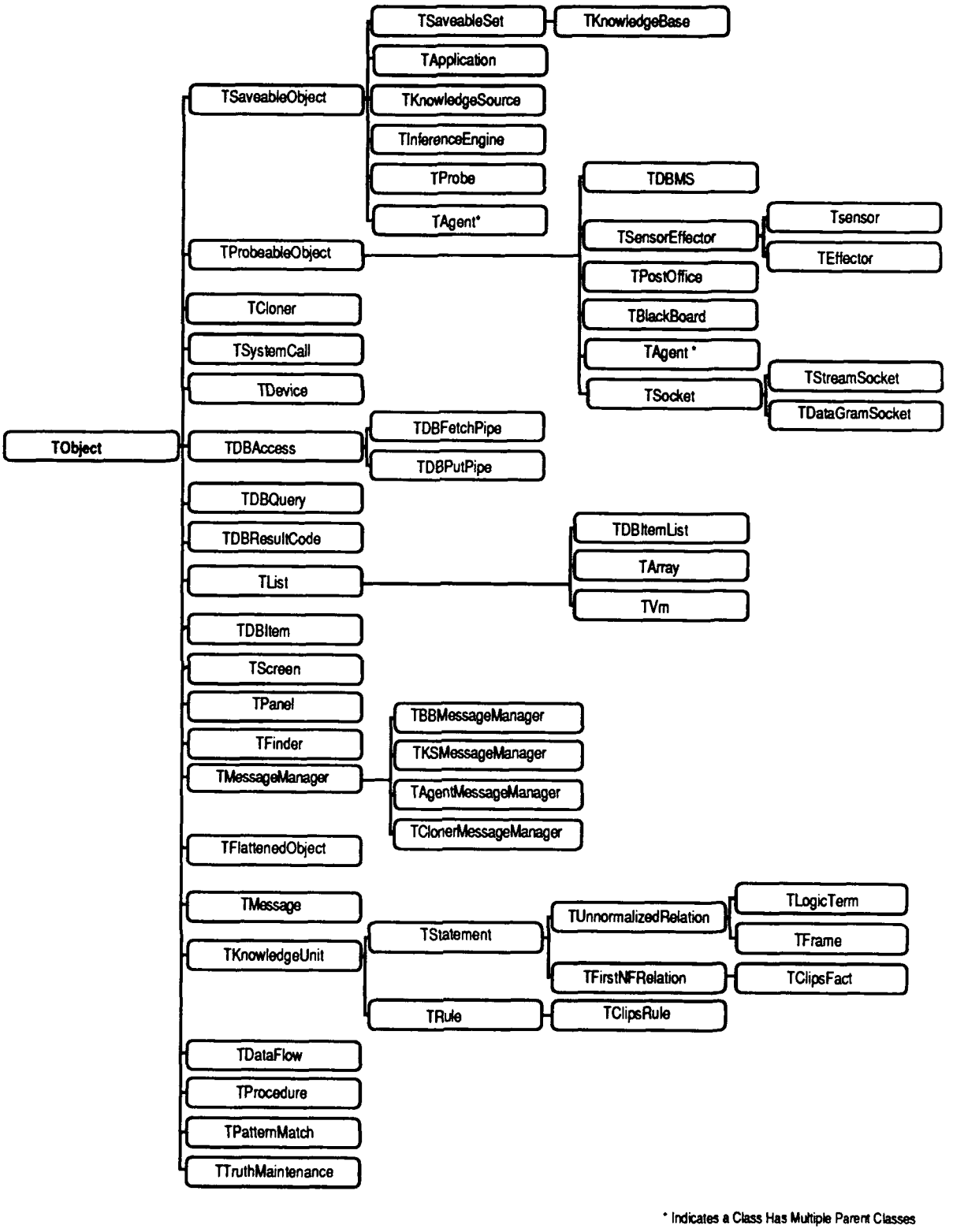


Figure 4.8 . The K-Bus Class Hierarchy

Each class shown is detailed in the following sections using a common description template. This is shown below with descriptions of each part. The classes have been given names beginning with the capital letter "T", following a convention used in the MacApp system, to make it easy to distinguish object class names from other words.

Class

ClassName

Description

A short description of the role of the object

Superclass

Immediate parent classes

Subclasses

examples of subclasses, not necessarily complete

Private Components

Components not directly accessible to objects outside of this object. Components can be data, objects, or references to objects.

Public Components

Components directly accessible to objects outside of this object. Components can be data, objects, or references to objects.

Functions

ActionName: iparm1 x iparm2 x ... x iparnN -> oparm1 x oparm2 x ... x oparnN

Action: Short description of the object function ActionName that takes the input parameters iparm1 through N and provides the output parameters oparm1 through N.

Policy

Formal definitions of objects and abstract data types (a subset of objects) include a set of assertions about correct application of the object class, called pre conditions, and a set of axioms that describe the semantics of the functions of that object. This level of formality is too extreme for this early stage of K-Bus design, and instead, the policy section of the object class description provides pre condition and operation axiom information in an informal prose.

Class

TObject

Description

The base class for all K-Bus Objects

Superclass

Subclasses

all other classes

Private Components

Public Components

Functions

Flatten : -> storableForm

Action: Creates a flat storable form of the object

UnFlatten : storableForm -> TSaveableObject

Action: Initializes the object with the storableForm

Delete : -> nil

Action: Deletes an object from memory

Display : -> nil

Action: Displays an object

Edit : -> nil

Action: Edits an object

Class

TSaveableObject

Description

Supplies generic services for persistent objects, never instantiated

Superclass

TObject

Subclasses

TSaveableSet, TKnowledgeSource, TAgent

Private Components

Contents
Collection of external forms (text, icon, picture ...)

Public Components

Identifier
PersistentCopyId

Functions

Retrieve : PersistentCopyId -> TSaveableObject
Action: Reads an object from persistent storage into memory
Store : -> nil
Action: Writes a flattened object in memory onto persistent storage using its PersistentCopyId

---

Policy

1) Identifier is for remote access and indexing objects in a set
2) PersistentCopyID is a database access path
3) Retrieve/Store/Delete will call layer 3 DBMS services and will interface with the location broker
4) Editing includes browsing; sets know how to access their members
5) The externalForm used in UnFlatten must have been Flattened from the same class.

Issues

1) Display and edit should have a parameter of which external form to display; maybe there's no cache of displays, just the methods
2) Editing transforms the image form, but may call Store to transform the persistent form ('save on disk')
3) Unflattening may involve instantiating a new TSaveableObject and replacing this object with this new object. This is the reason for the output TSaveableObject, which may or may not be the same object to which the method call is addressed.

Class

TSaveableSet

Description

Supplies generic services for sets of objects, never instantiated

Superclass

TObject

Subclasses

TKnowledgeBase

Private Components

Set of TSaveableObjects

Functions

RetrieveMember : TSaveableObject.Identifier-> TSaveableObject|nil

Action: Accesses a member object, or returns nil if no such member

InsertMember : TSaveableObject -> nil

Action: Inserts an object into the set, updating the indexing scheme

DeleteMember : TSaveableObject -> nil

Action: Deletes an object from the set, updating the indexing scheme

---

Policy

1) A set is assumed to be unordered with a private indexing scheme

Issues

1) RetrieveMember is passed the identifier by some query pre-processor that translates a condition into an identifier

Class

TProbeableObject

Description

An abstract class providing basic probe attachment support to its subclasses. Overrides are required to realize working probe attachment.

Superclass

TObject

Subclasses

Private Components

Collection of attached Probes

Public Components

Functions

attachProbe : Probe ->

Action: Installs a probe in the object, sets reference to its TPostOffice in the probe

detachProbe : ProbeID ->

Action: Removes the probe from the object.

enableProbe : ProbeID ->

Action: Puts the probe in an active monitoring state, where it can recognize its trigger condition.

disableProbe : ProbeID ->

Action: Puts the probe in a passive, non-monitoring state, where the probe cannot fire.

showProbes : -> TList

Action: Provides a collection of TProbe currently attached.

Policy

1) The probeable object must interpret the probe trigger and event information to realize the event pattern to action translation. Support within this abstract class may be added if support for the unification process appears shareable among the object subclasses.

Class

TList

Description

Provides an key indexible collection of objects

Superclass

TObject

Subclasses

Private Components

TObject list

Public Components

Functions

Count: -> Integer

Action: Returns the number of items in the array

Get: Integer -> TObject

Action: Retrieves the object at index Integer

Put: Integer x TObject ->

Action: Retrieves the object at index Integer

Delete: TObject ->

Action: Searches the list for the first reference to the TObject and removes it from the list

DeleteAll: ->

Action: Deletes all the items from the list (doesn't free the objects)

Each: <Function>(TObject) ->

Action: Applies the function to each object in the array

First: -> TObject

Action: Retrieves the first item in the list

FirstThat: <Boolean Function>(TObject) -> TObject|nil

Action: Calls the function for each item in the list until it returns TRUE, then returns that object. Return nil if the function is never satisfied.

InsertFirst: TObject ->

Action: Put the TObject item at the front of the list

InsertLast: TObject ->

Action: Put the TItem at the end of the list

Class

TFlattenedObject

Description

The flattened for (storable form) of an object

Superclass

TObject

Subclasses

everything

Private Components

Public Components

ObjectType
ObjectIdentifier
flattenedString

Functions

Policy

1) A basic envelope structure for flattened objects. All objects themselves know how to unflatten themselves from a flattenedObject, and to flatten themselves into one.

4.3.2 Preliminary Design Specification

4.3.2.1 Devices & Users Layer Design Specification

This layer is not specified in this report, as it represents the given external platform and real world interfaces upon which the architecture must rest, and therefore is more a part of the problem space than the solution space specified in this report. The K-Bus architecture is intended to support highly flexible and portable applications on many platforms, and in many application situations. The power of the hardware platforms, and the bandwidth of its communication channels will constrain the particular application set that can be hosted, and the efficiency of some of the K-Bus functions, but should not constrain application of the logical architecture.

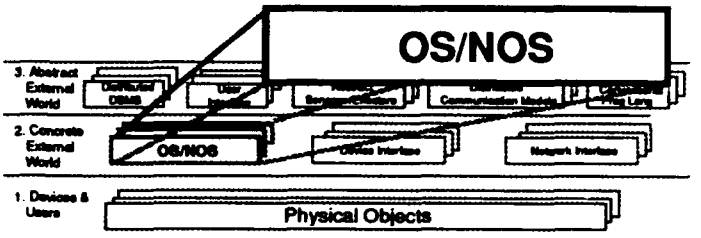
4.3.2.2 Concrete External World Layer Design Specification

4.3.2.2.1 OS/NOS Related Objects

The interface to the underlying operating system is realized with a single object with separate methods for each operating system call. The calls are patterned after UNIX calls. Additions may be needed for real time operations.

Class

TSystemCall



Description

Abstract class for underlying OS calls

Superclass

TObject

Subclasses

Private Components

Communication area
Result Codes

Public Components

Functions

<OSCallName>: <parameters> x [<errorHandler>]-> resultCode

Action: Invokes the OS call <OSCallName> passing the parameters specified. The errorHandler function is optional, and will be invoked if other than a normal return results

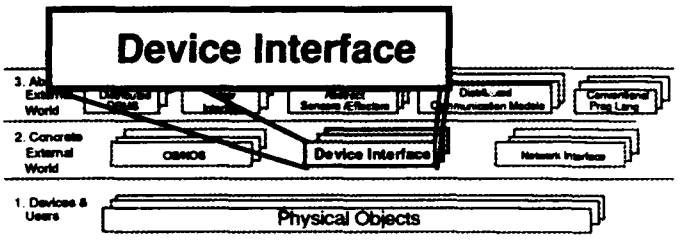
Policy

1) This will be elaborated to a specific set of OS calls.

4.3.2.2.2 Device Interfaces

Class

TDevice



Description

An object view of the underlying device driver

Superclass

TObject

Subclasses

Private Components

fileDescriptor
pathName

Public Components

Functions

TDevice: pathName x flags x modes -> resultCode
Action: object constructor attaches the object to the underlying device driver, allocates and initializes necessary support structures.

~TDevice : address ->
Action: destructor: severs connection with underlying device driver and deallocates support structures.

read : buffer x count -> number
Action: reads up to count bytes from the device into buffer. Number indicates the actual number of bytes obtained.

write: -> buffer x count -> number
Action: writes up to count bytes from the buffer. Number indicates the number of bytes written.

ioctl : command x arg -> ErrorCode
Action: Catch all command call for device specific commands. Arg can point to input or output structures.

fd : -> fileDescriptor
Action: get the associated fileDescriptor

path : -> pathName
Action: get the associated pathName.

Policy

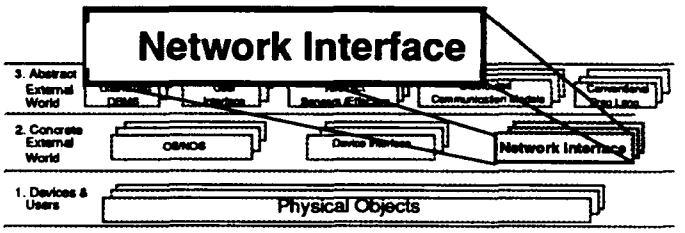
1) Patterned after UNIX device driver interfaces, where devices look like files in most respects.
2) The file descriptor is made available to allow access to streams.

4.3.2.2.3 Network Interfaces

Let the basic network interface paradigm follow the BSD 4.3 socket approach and build on this for other communication schemes. The object definitions that implement this are purposely kept very similar to the UNIX socket interfaces. The implementation is not constrained to use the actual underlying socket mechanisms.

Class

TSocket



Description

Basic abstract socket object

Superclass

TProbeableObject

Subclasses

TStreamSocket, TDatagramSocket

Private Components

SocketDescriptor
SocketAddress

Public Components

Functions

bind : address -> ErrorCode
Action: associates a logical address with a socket

connect : address -> ErrorCode
Action: initiates a connection to a socket, address is the logical address of the target socket

getSockName : -> address
Action: returns the current logical address for the socket

setSockOpt : level x optname x optval -> ErrorCode
Action: set options associated with a socket

getSockOpt : -> level x optname x optval
Action: get options associated with a socket

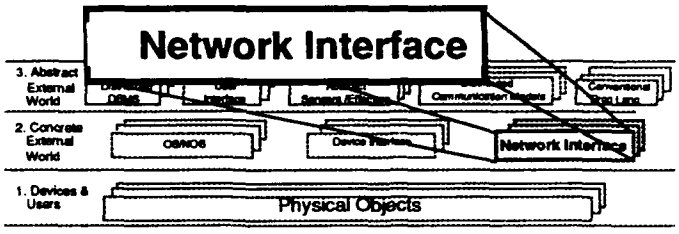
getPeerName : -> address
Action: returns the current name for peer connected to the socket

Policy

1) Follows the UNIX socket rules

Class

TStreamSocket



Description

Socket type supporting virtual circuit connections

Superclass

Socket

Subclasses

Private Components

Public Components

Functions

listen : queueLength -> ErrorCode
Action: associates a logical address with a socket

accept : address -> address x ErrorCode
Action: initiates a connection to a socket, address is the logical address of the target socket

send : message -> INTEGER x ErrorCode
Action: transmit a message to the connected socket and return the number of characters sent

sendAside : message -> INTEGER x ErrorCode
Action: transmit an "out of band" message to the connected socket and return the number of characters sent

recv : -> message x INTEGER x ErrorCode
Action: receive messages from a connected socket and remove from queue

peek : -> message x INTEGER x ErrorCode
Action: receive messages from a connected socket and do not remove from queue

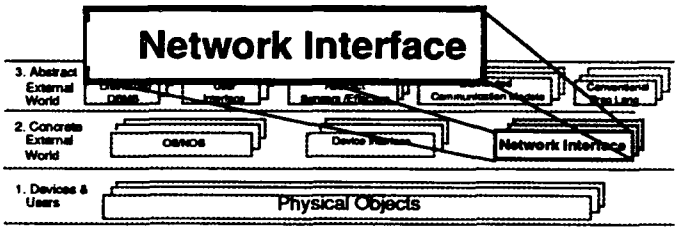
recvAside : -> message x INTEGER x ErrorCode
Action: receive "out of band" messages from a connected socket

Policy

1) Follows the UNIX socket rules

Class

TDataGramSocket



Description

Socket type supporting datagram communications

Superclass

Socket

Subclasses

Private Components

Public Components

Functions

sendTo : message -> INTEGER x ErrorCode

Action: transmit a message to the socket and return the number of characters sent

sendAsideTo : message -> INTEGER x ErrorCode

Action: transmit an "out of band' message to the socket and return the number of characters sent

recvFrom : -> message x address x INTEGER x ErrorCode

Action: receive messages from a socket and remove from queue

peekFrom : -> message x address x INTEGER x ErrorCode

Action: receive messages from a socket and do not remove from queue

recvAsideFrom : -> message x address x INTEGER x ErrorCode

Action: receive "out of band" messages from a socket

Policy

1) Follows the UNIX socket rules

2) Address contains the source address of the message

4.3.2.3 Abstract External World Layer Design Specification

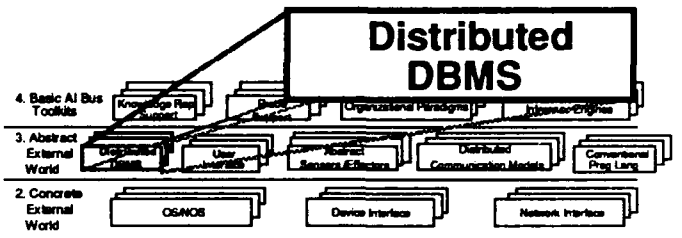
4.3.2.3.1 Shared Objects

4.3.2.3.2 Distributed DBMS

The DBMS at layer 3 is modeled as a single object with methods which are very close to those made available though the program interface. Each process thread is assumed to have its own DBMS representative through which it deals with the distributed DBMS, and which hides the distributed nature of the Database. Higher level objects in layer 4 may be useful to model an object oriented database on top of the relational one represented by these objects. The DBMS object classes can also be used to access the underlying file system regarded as a simple reduced functionality database.

Class

TDBMS



Description

An object class representing a logical DBMS, either stand-alone or distributed.

Superclass

TProbeableObject

Subclasses

Private Components

Stored Data

Public Components

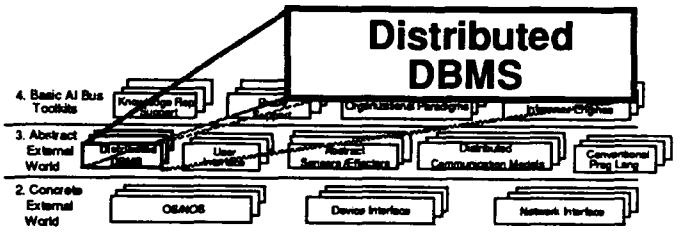
Functions

Policy

1) *The object's constructor establishes the DBMS/object link. The constructor ndicates if this is a full capability DBMS, or a flat file system.*
2) *Multiple TDBMS instances can exist, even in a single process, and correspond to something similar to ORACLE logon's.*

Class

TDBAccess



Description

Provides basic relational SQL Query Access to the distributed DBMS

Superclass

TObject

Subclasses

Private Components

DBMS Communication Areas

Public Components

Functions

SetUp : DBQuery x DBMS ->

Action: Initializes information necessary to do an attach. This does not make a DBMS call; it is a constructor addendum, and might even become a part of the constructor.

Execute : -> DBAccessResult

Action: Connects to the DBMS and executes the query

Abort :

Action: Rolls back the actions of the query, if possible with the query type..

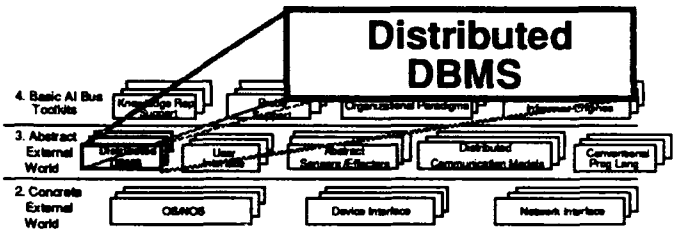
Policy

1) Assumed can be used for basic queries such as deletes, modifies, etc. that do not return a value (are non fetch type). Use the DBFetchPipe and DBPutPipe for doing multiple inserts and for retrievals.

2) If a flat file system, the query is limited a single relation reference, where the relation corresponds to the schema of the file records. Further implementation imposed limits for files may be required.

Class

TDBFetchPipe



Description

Supplies a pipe like query access to the Distributed DBMS

Superclass

TDBAccess

Subclasses

Private Components

Public Components

Functions

Attach : TDBMS -> DBAccessResult

Action: Connects to the DBMS and establishes query

Next : -> DBResultCode x DBItemList

Action: Retrieves next item from the DBMS as per the query. The result is placed via access method calls as per the DBItemFrame.

Next : count -> DBResultCode x DBItemList

Action: Retrieves next count item(s) from the DBMS as per the query. The result is placed via access method calls as per the DBItemFrame.

Detach :

Action: Detaches the fetch pipe from a DBMS session.

Policy

1) User assumed to have provided consistent SQL fetch query and done a setup prior to attachment.

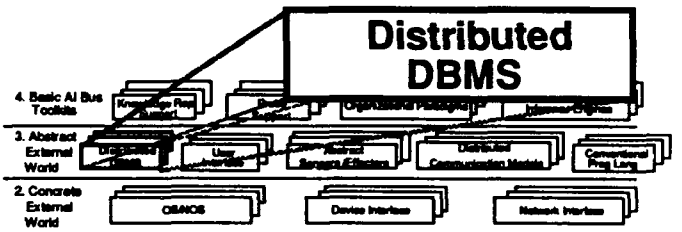
2) The DBItemList provides the information concerning the query results, which are placed in the frame when the query occurs. This list includes necessary type and size information required by the DBMS output conversion functions.

3) If a count is provided, it controls the number of items the method attempts to fetch in the single call.

4) Corresponds to an ORACLE cursor, in many ways.

Class

TDBPutPipe



Description

Supplies a pipe like store access to the Distributed DBMS

Superclass

TDBAccess

Subclasses

Private Components

Public Components

Functions

Attach : TDBMS -> DBReultCode

Action: Connects to the DBMS and establishes query

Put : DBItemList -> DBResultCode

Action: Puts the items in the itemList into the DBMS through the item by item pipe.

Detach :

Action: Detaches the put pipe from a DBMS session and confirms the transaction.

Abort :

Action: Detaches the put pipe from a DBMS session and rolls back the transaction, undoing all actions since the attach.

Policy

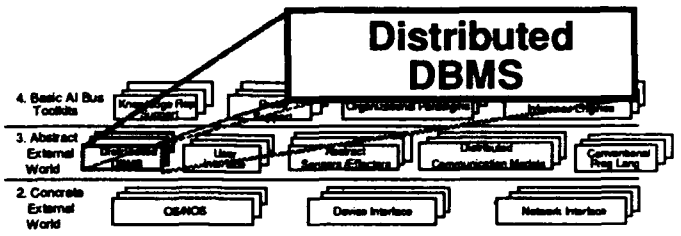
1) User assumed to have provided consistent SQL insert query and done the setup prior to attachment.

2) The DBItemList provides the information concerning the query results, which the caller is assumed to have built prior to doing any Put operations. The ItemCount in the itemList is used to determine the number of items to place in the database.

3) Corresponds to an ORACLE cursor, in many ways.

Class

TDBQuery



Description

An Acceptable query for the underlying DBMS (expected to be SQL)

Superclass

TObject

Subclasses

Private Components

A Copy of the Query String

Public Components

Functions

SetQuery : String -> nil

Action: Sets the internal copy of the query to the string.

Policy

1) Currently, expected to be just a container for the ASCII SQL Query, although it might also serve as the container for the compiled version.

2) Queries for flat file access may be implementation limited.

Class

TDBResultCode

Description

Indicator of results from a DBMS access call

Superclass

TObject

Subclasses

Private Components

The value of the result code returned from a DBMS call
DBMS

Public Components

Functions

ErrorMessage : -> String

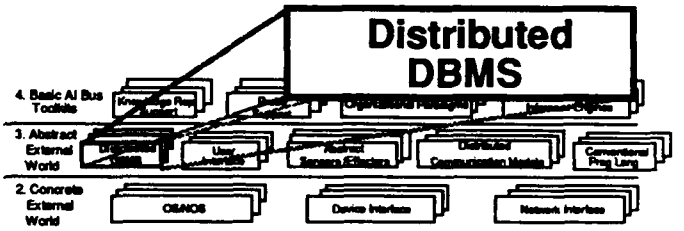
Action: Returns the error message text corresponding to the result code.

Error: -> Boolean

Action: Returns true if the result code is an error.

Class

TDBItemList



Description

The object that can receive the single or multiple tuple results from a Distributed DBMS query or supplies the values for a DBMS insert operation.

Superclass

TList

Subclasses

Private Components

ItemCount

Public Components

Functions

SetItemCount: count ->

Action: Sets the number of items represented by the itemList

ItemCount: -> count

Action: Returns the number of items represented by the itemList

Policy

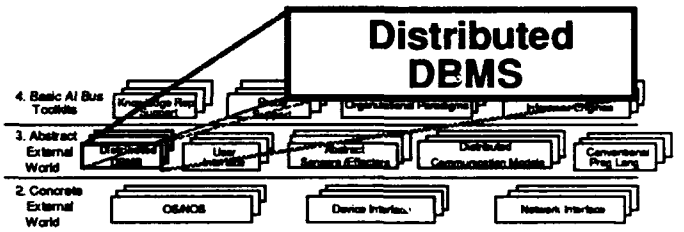
1) May be able to simply use a TList of items

2) The DBFetchPipe creates one of these objects for every next call. The user should dispose of it if just using it for an envelope.

3) The position in the list maps into the position of the field or expression in the select list of the related SQL statement

Class

TDBItem



Description

Describes a single item/field resulting from a DBFetchPipe Next operation, or being prepared for a DBPutPipe operation

Superclass

TObject

Subclasses

Private Components

ItemInternalType
ItemExtnlType
ItemSize
ColumnName
ValueArray

Public Components

Functions

ValueArray: -> TArray
Action: Returns an array object containing the values of the items
SetValueArray: TArray ->
Action: Sets the array object containing the values of the items
Name : -> String
Action: Returns the column name of the item
IType : -> Integer
Action: Returns the internal type of the item using the DBMS type codes
EType : -> Integer
Action: Returns the external type code of the item
SetEType : x Integer -> nil
Action: sets the external type for the item.
Size : -> Integer
Action: Returns the size of the item
SetSize : Integer -> nil
Action: Sets the size of the item
Size : -> Integer
Action: Returns the size of the item

Policy

1) The TArray object is access using its methods to retrieve the individual values.

Class

TArray

Description

Provides an indexible array of objects, indexed by integer from 1 to the count of objects stored

Superclass

TList

Subclasses

Private Components

TObject list

Public Components

Functions

Count: -> Integer

Action: Returns the number of items in the array

Get: Integer -> TObject

Action: Retrieves the object at index Integer

Put: Integer x TObject ->

Action: Retrieves the object at index Integer

Each: <Function>(TObject) ->

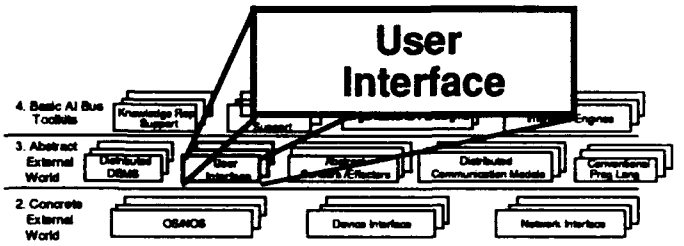
Action: Applies the function to each object in the array

4.3.2.3.3 User Interface

The user interface objects are designed to provide a program independent direct manipulation interface. This allows a consistent user interface to be created for the variety of applications envisioned for the K-Bus, yet relieves application developers of most of the details of constructing the interface and following strict user interface guidelines. The classes specified provide a objects that will deliver user actions to the applications in a consistent way, independent of whether the user selects from a menu, or types a response. The current definition follows the NASA TAE system very closely. Knowledge of TAE will be necessary to understand the detailed discussion of these objects.

Class

TScreen



Description

Represents a connection to a display server.

Superclass

TObject

Subclasses

Components

PanelList

Functions

TScreen : displayName -> XrDisplayId

Action: Initializes communication with the display server, and initializes related library code for further use.

nextEvent : -> panelEvent

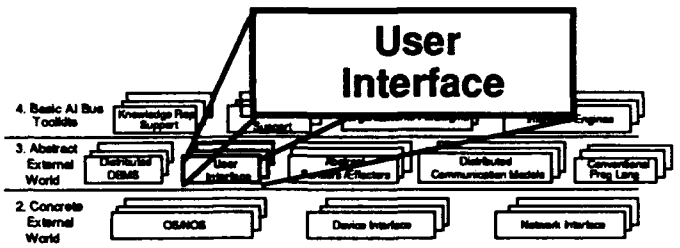
Action: Blocks until a panel related event occurs, then returns the event record for that event.

Policy

1) *Currently restricted to a single physical screen.*
2) *TAE documentation provides alternatives for blocked wait for next event. Behavior of nextEvent may be inappropriate if Xr objects beyond those from TAE are used.*

Class

TPanel



Description

Contains one or more items which provide the mechanism for interaction between the user and the application program.

Superclass

TObject

Subclasses

Components

ItemList

Functions

TPanel : dataVm x viewVm x Window x eventRec x flags -> TPanel

Action: Constructor for instantiating a TPanel object. The dataVm and viewVm objects must exist prior to instantiating a TPanel. The window provided specifies the relative window for the panel. The flags indicate whether the new panel is a child of the root or the relative window.

DisplayBusy : -> resultCode

Action: Displays an indicator implying that the program driving the panel is busy. Ignores User Events while busy.

StopBeingBusy : -> resultCode

Action: Terminates the busy condition established by DisplayBusy.

CloseItems : -> resultCode

Action: Completes the user input for panel items and merges the user input into the target Vm object for use by the application..

~TPanel : -> resultCode

Action: Erases the panel from the screen and frees up the associated memory. It does not free up the target or view Vm objects.

Reset : -> resultCode

Action: Resets the panel to its initial values supplied in the constructor

Message: message -> resultCode

Action: Generates a message box for the panel and displays the text of the message in it, and waits for acknowledgement.

ParmReject: name x message -> resultCode

Action: Generates a message box for the panel and displays the text of the message in it, and waits for acknowledgement. It then reverts back to the previous parameter value.

ParmUpdate: name -> resultCode

Action: Updates the display for a particular parameter on this panel. The target Vm must be updated prior to calling this function.

ViewUpdate: name x newView x viewParm-> resultCode

Action: Updates the view of the particular parameter using the viewParm of the newView. Used to update icons and static text on displays. Ignored for other data types.

WindowID : -> XWindowId

Action: Returns the X window Id of the panel

XrPanelId : -> XrPanelHandle

Action: Returns the Xr defined panel handle of the panel

Policy

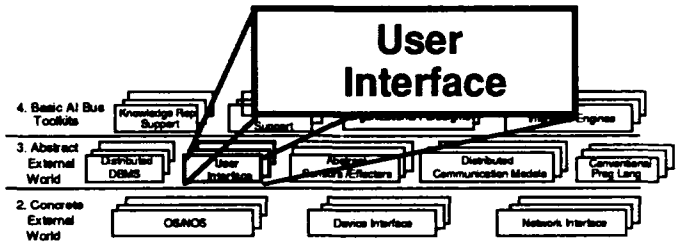
1) The TScreen server object must have been instantiated prior to the constructor being called for a panel.
2) After the constructor call, the panel is active and awaiting user input. The caller is not blocked.
3) The dataVm and viewVm must match a predefined resource file format.
4) See the TAE documentation for explanation of the terms used

Class

TVm

Description

Dynamic Memory Variable collection. Used in conjunction with the user interface objects.



Superclass

TList

Components

Functions

TVm : abortFlag -> TVm

Action: Creates an initially empty Vm object instance. If the abortFlag is set, errors will result in a process abort (with message), otherwise, caller program will be informed at later calls to the Vm object about errors.

Find : name -> TVariable

Action: Returns an object containing an individual variable from the Vm object. Aborts if abortFlag set and the name is not found.

FindVar : name -> TVariable

Action: Returns an object containing an individual variable from the Vm object. Ignores the abort flag and returns nil if the name is not found.

GetAttributes: name -> type x n x defaultFlag x accessMode

Action: Extracts the attributes for a particular variable. Type can be V_INTEGER, V_REAL, V_STRING, and V_NAME. The count, n, indicates the number of values for that variable. The defaultFlag indicates if the value is the default. The accessMode may be V_IN, V_OUT, V_INOUT, or P_NONE (no access mode).

Read : spec -> returnCode

Action: Reads a saved parameter file and stores the information in the Vm object. Spec is a host file specification of the file to be read.

Write: spec -> returnCode

Action: Writes a saved parameter file using the information in the Vm object. Spec is a host file specification of the file to be written.

Set: name x count x valArray x mode -> returnCode

Action: Sets the values for integer, real, or string* (overloaded function). These are set from an array of integers, reals, or pointers to strings.

SetMin: name x min -> returnCode

Action: Sets the minimum vector count of the variable. Has no effect if > the current count.

SetMax: name x max -> returnCode

Action: Sets the maximum vector count of the variable. Has no effect if < the current count.

SetParmPage: name x flag -> returnCode

Action: Sets the page indicator of the variable, where the flag is true or false.

SetStringLength: name x strlen -> returnCode

Action: Sets the maximum length of a string or keyword parameter

SetValid: name x count x vlow x vhigh -> returnCode

Action: Sets the low and high valid values for real and integer variables, where vlow and vhigh are real or integer arrays of length count.

SetValid: name x count x Text* -> returnCode

Action: Sets the valid values of the string variable

4.3.2.3.4 Abstract Sensors/Effectors

These objects provide a level of access higher than that provided by the basic device drivers of layer 2. The sensors and effectors are basic transducer objects, providing conversion between the signal and symbol world. The goal of the layer 3 objects is to provide building blocks to construct efficient, yet very understandable abstractions of the transducers with high reusability. Implementation may be pushed clear to layer 1 for efficiency. For very "intelligent" sensors and effectors, the layer 3 objects may be simply their "software window".

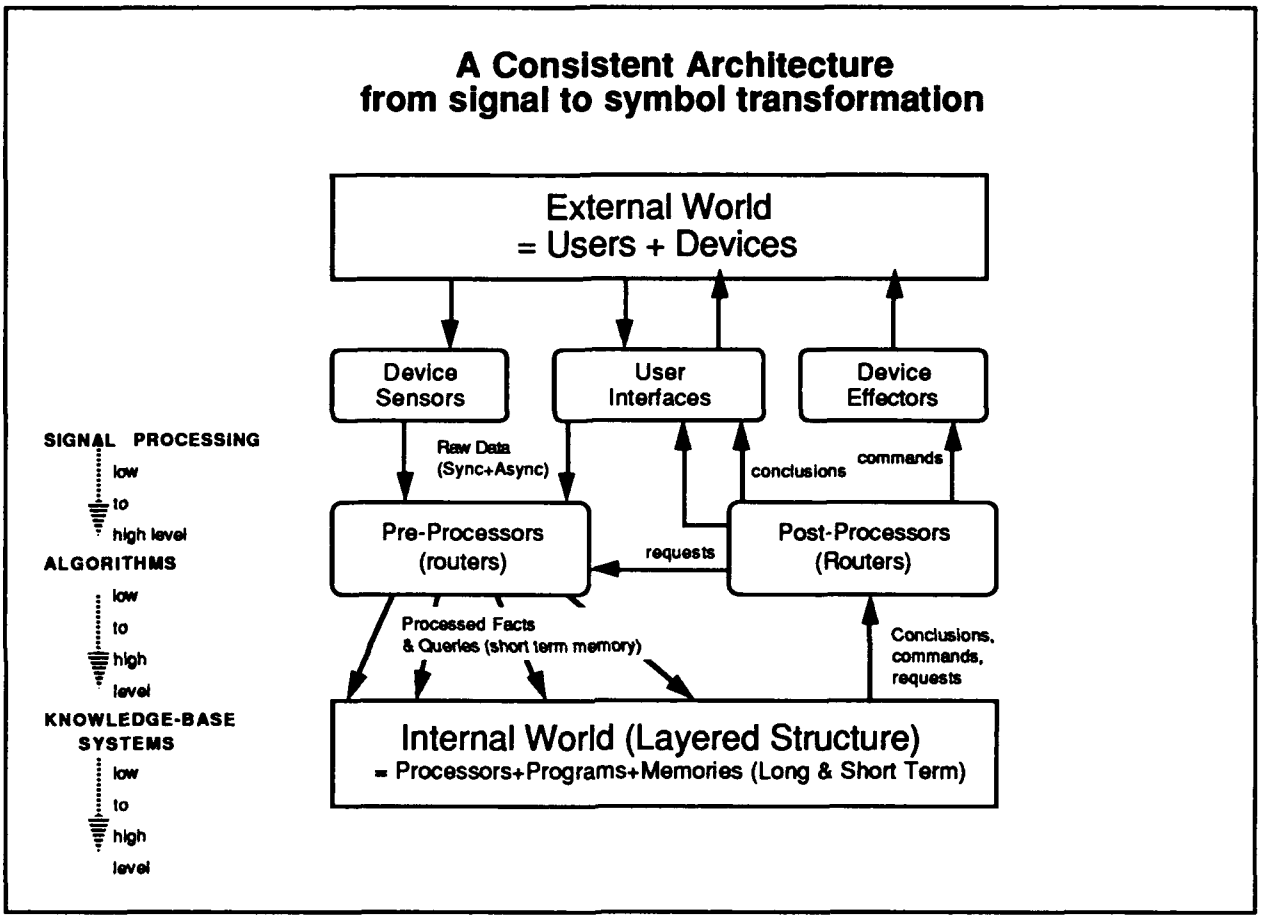


Figure 4.9. Lower Layers Communicate With The Physical World

Desiderata: Since sensor and effector loops can be very time critical, it would be nice to be able to specify something like a "block diagram" of components that can react without intervention at the higher level, and provide the final symbol output to the higher level components of a subsystem. Calls would prove a way for the caller to declare the types and parameters of the blocks, and the interconnections. Methods would then allow control over the blocks, as well as access to their final outputs. Idea sources for this include:

1) ATE languages such as ATLAS

2) Constraint programming systems

3) Analog computer simulators

4) Instrumentation systems currently available for PCs

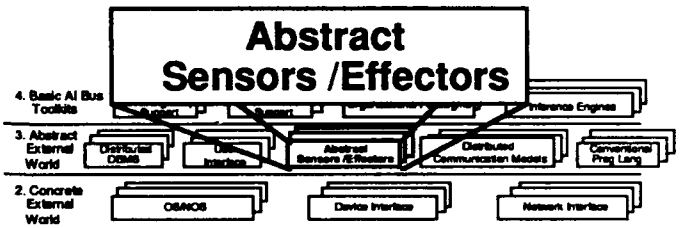
5) Current process control literature

6) Streams

One approach, following the OOP tradition would be to generalize the block diagram for sensors and effectors, and provide for methods to support each box on the block diagram. A toolkit to support declaration and construction of this type of abstract sensor/effector would be needed. This toolkit could make use of a direct manipulation interface to allow designers to manipulate block diagram components to describe a sensor/effector process. The toolkit would generate the corresponding object oriented code.

Class

TSensorEffector



Description

Abstract class for abstract sensor/effector support

Superclass

TProbeableObject

Subclasses

TSensor, TEffector

Private Components

related TDriver objects
attached probe support information

Public Components

Functions

reset : ->
Action: initializes the abstract sensor/effector. Overridden by specific implementations. The default reset is no action.

getState : -> SensorEffectorState
Action: provides internal state information

setup : ->
Action: carries out actions to set up the measurement/command process

start : ->
Action: carries out actions based on the setup. This may establish a cyclic measurement or control loop depending on the particular sensor/effector.

halt : ->
Action: Terminates any continuous measurement or control process associated with the abstract sensor/effector..

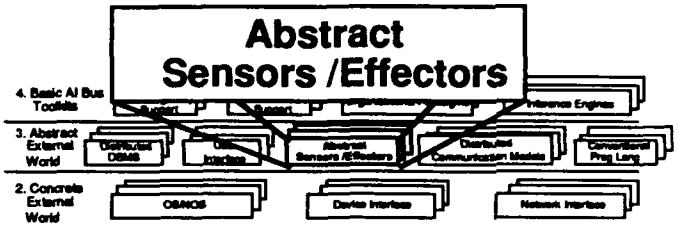
Policy

1) Provides transducer services beyond that provided by the driver objects themselves. Could be used to construct virtual sensors with data sources from multiple physical sensors.

2 Sensors are passive in terms of communicating their readings. Probes can be used to make them active, or they can be queried for their readings directly. Any probe attached must have a SensorEventTrigger.

Class

TSensor



Description

Abstract class for abstract sensor/effector support. The basic structure in which one "builds" an abstract sensor.

Superclass

TSensorEffector

Subclasses

Private Components

SensorState

Public Components

Functions

read : -> TObject

Action: provides the next reading object from the last initiate. Returns nil if there is no unread measurement.

initiate: ->

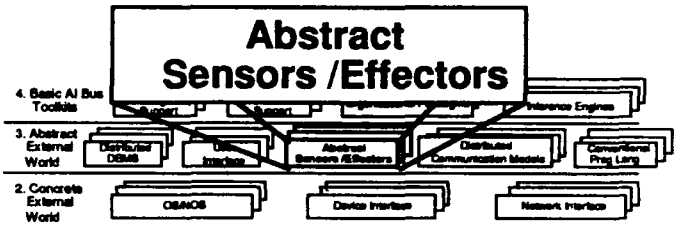
Action: enables a measurement operation by the sensor function and to cause the measured values to be taken and held for subsequent access.

Policy

1) Measurements may be carried out by an independent process established by the start method, or may be initiated by the initiate method. Use of both together may cause interference and is dependent on the particular sensor type. Initiate forces a single reading to be taken, while start establishes a "continuous "sampling loop.

Class

TEffector



Description

Class for abstract Effector support (actuator support). The basic structure in which one "builds" an abstract effector.

Superclass

TSensorEffector

Subclasses

Private Components

EffectorState

Public Components

Functions

Set : subUnitID x value ->

Action: Sets the particular subunit to the value indicated. Used to set up command parameters.

Command : subUnitID x commandValue ->

Action: Queues the particular command Value to the particular subunit. The command itself is not actually issued to the subunit until a start is sent.

Policy

1) Commands are set up, then initiated through the start method.

2) Toolkits to build effectors will be specified as the architecture is elaborated further.

4.3.2.3.5 Distributed Communications Models

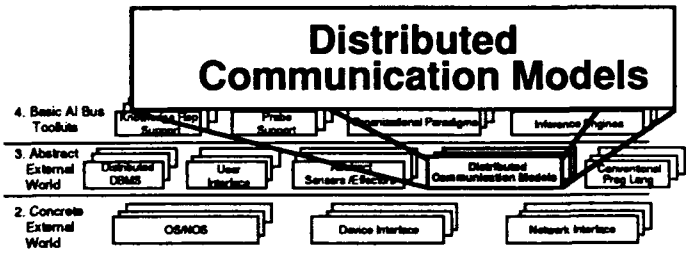
These objects form the building blocks used to provide distributed problem solving paradigms. The building blocks are necessarily low level, to provide flexibility in constructing communication schemes that fit the problem solving paradigm, yet provide a level of isolation from the underlying level 2 network dependent services.

Location transparency is realized using these objects, but object location decisions and bootstrap instantiation decisions must be handled by some entity that is location aware. This could be through off line decision making resulting in generation of permanent residence decisions, or could be dynamically made by location broker entities.

Objects such as agents and knowledge sources of level 4 can be made remote and provided their own process thread by using subclasses of the TMessageManager class of this level.

Class

TCloner



Description

Reconstitutes active objects from stored or flattened form, and invokes it. Primarily intended as a remote service for instantiating active objects, such as bootstrapping application suites.

Superclass

TObject

Subclasses

Private Components

Public Components

Functions

ReadObject : objectClass x object.Identifier -> nil
Action: accesses the stored flattened object and reconstitutes

Unflatten Object : objectClass x flatObject -> object.Identifier
Action: searches its directory for a remote object with a particular name

Create Object : objectClass x initParms -> object.Identifier
Action: creates the object from scratch using the initparms provided.

Policy

1) A TCloner (with associated TPostOffice) is bootstrapped on every platform that allows remotely instantiated active objects. These TCloner active objects are registered with a TFinder server that allows access by specific platform, as well as possible other attributes.

2) The TCloner must know of the active object Class in order to be able to instantiate it. The ReadObject and UnflattenObject services are limited to objects that inherit from TStoreableObject.

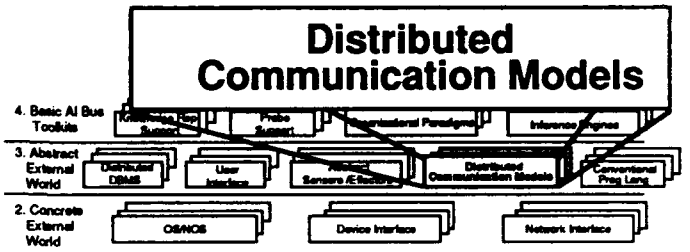
3) ReadObject knows where to look for stored objects, possibly based on the object identifier.

4) TCloner uses the flattening and unflattening self knowledge of the TStoreableObjects.

5) Resulting objects are registered with the name server.

Class

TPostOffice



Description

Handles message collection, shipping, and distribution. A TPostOffice object will exist for every active object providing routing of messages by global name, and queuing of input messages with selective access.

Superclass

TProbeable Object

Subclasses

Private Components

Collection of incoming messages (TMessage)

Public Components

Functions

Send : TMessage ->

Action: forward the message to the receiver indicated in the message.

Check : attributePattern -> integer

Action: provide the count of queued messages meeting the attribute pattern.

Receive: attributePattern -> TMessage|nil

Action: return the first message received that meets the attributePattern, or nil if none meet the pattern.

Wait: attributePattern x maxwait -> TMessage |nil

Action: suspend until a message matching the attributePattern is available, return the first message received that meets the attributePattern, or nil if none meet the pattern after maxwait time.

Policy

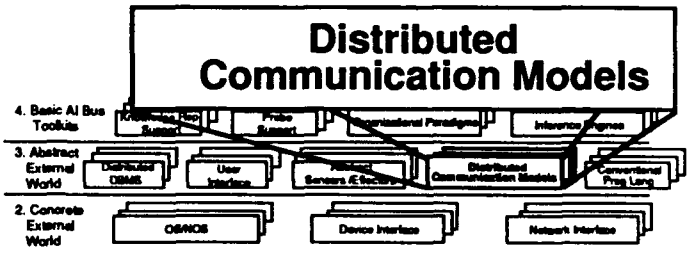
1) Messages are delivered FIFO within the group matching an attributePattern.

2) Message structure includes sender and receiver and this information must be set in the TMessage object prior to sending it.

3) Wait provides a low overhead method of suspending until there is work to do. Special aborts could be handled by returning a special TMessage, so clients waiting must check that the received TMessage is not one of the special result messages.

Class

TMessage



Description

An abstract structure within which messages transported through the TPostOffice are cast.

Superclass

TObject

Subclasses

Private Components

sender.Identity
receiver.Identity
attributeCollection
flattenedObject

Public Components

Functions

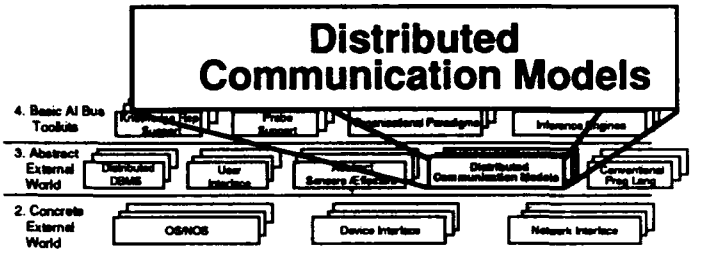
None

Policy

1) Sender and receiver identities are those registered with the TFinder services.
2) The flattenedObject type and structure is assumed understood by the receiver based on information found in the attributeCollection, or the header information of the flattenedObject.
3) Emulation of remote method calls is done by putting the particular method id as the attribute, and the parameters within the flattened object.

Class

TFinder



Description

A remote object name server that provides address information necessary to address communication to a remote object. This object interfaces with the name server processes. Can be used as the basis for higher level broker objects.

Superclass

TObject

Subclasses

Private Components

LocationDirectory

Public Components

Functions

Register : objectName x objectAddress x attributes -> nil

Action: adds a remote object to the TFinder's list of remote objects

AddressOf : objectName -> objectAddress x attributes | nil

Action: searches its directory for a remote object with a particular name

Find : attributePattern -> addressList

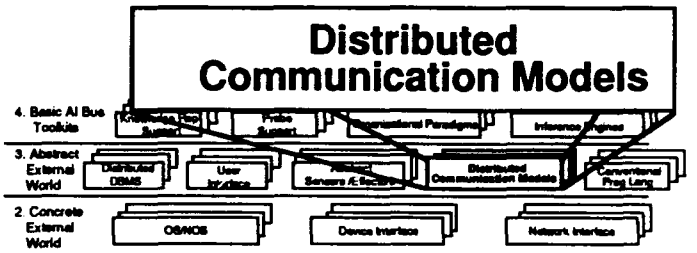
Action: searches its directory for a remote object with attributes matching the attributePattern. The attributePattern is a text string with some wildcard capability.

Policy

1) *The attributes are a simple collection of text string objects, representing key words describing the attributes of the remote object.*
2) *The addressListObject returned is a collection of objectAddress x attributes that match the attributePattern. Normal collection access methods can then be used to get to specific entries.*
3) *Remote objects must be registered to be found by others if their information is not explicitly known. TFinder acts like a distributed database of names and related information. Registered objects will be known globally on the network.*
4) *Persistent stored objects may be represented in TFinder, with an indicator showing their stored state. This will not be the only directory for stored objects, but may be used for those that are referenced often.*

Class

TMessageManager



Description

A message to method call transducer object abstract base class. Used to make objects remote, in particular TAgent, TBlackboard, TKnowledgeSource, and TCloner objects.

Superclass

TObject

Subclasses

TBBMessageManager, TKSMessageManager, TAgentMessageManager, TClonerMessageManager

---

Components

TPostOffice

Functions

Invoke : ->

Action: Initially gives control to the message manager

Send : Message x Destination -> nil

Action: Sends message to the destination object

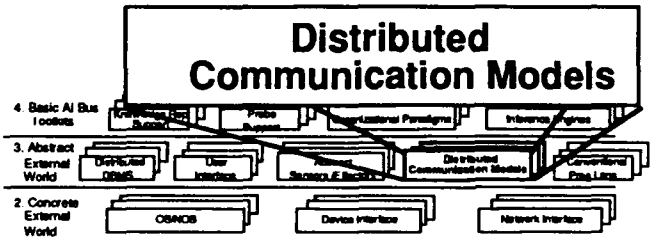
Policy

1) Uses the post office to serialize concurrent requests etc. and in addition contains a message dictionary to invoke its associated object's methods

2) May use a TFinder object to resolve Send's destination parameter, which will probably be the associated source's name for the target, e.g. an agent calling another by name. The destination may in fact be translated into the name of the target's message manager, if it has one.

Class

TClonerMessageManager



Description

A message to method call transducer object for TCloner class objects.

Superclass

TMessageManager

Components

Functions

Policy

4.3.2.3.6 Conventional Programming Languages

Although intended more as a placekeeper, Objects here might represent standardized ways of interfacing with "foreign functions", i.e., programs written in languages other than the chosen K-Bus implementation languages. This interface could provide the "glue" needed to convert objects to/from forms required by the "foreign" language.

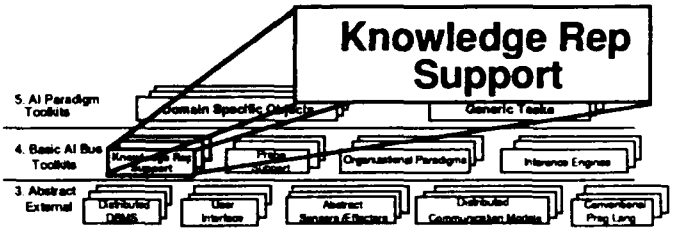
Hybrid languages (C++, Common Lisp w/CLOS) provide both objects and conventional representations. We'd like to be able to hand objects across the object interface between languages (implying some kind of glue, or flattening/unflattening w/common flattened form).

4.3.2.4 Basic K-Bus Toolkit Layer Design Specification

4.3.2.4.1 Knowledge Representation Support Specification

Class

TKnowledgeUnit



Description

Superclass for knowledge representation units, never instantiated

Superclass

TObject

Subclasses

TStatement, TRule, TFrame, TScript, ...

Components

Description
Rationale

Operators

<TKnowledgeUnitSubclass> : <Subclass> -> <Subclass>

Action: Coerces the source type to target type. Used for translation between heterogeneous knowledge representations

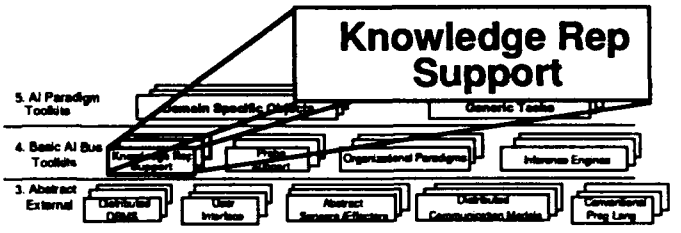
---

Policy

1) The Description and Rationale are external forms, for use in debugging and explanation

Class

TKnowledgeBase



Description

Indexed collection of knowledge units, a module (or context) of reasoning

Superclass

TSaveableSet

Subclasses

TClipsKB, TPrologKB, ...

Components

Set of TKnowledgeUnits

Policy

1) All units should be compatible (i.e be interpretable by the same inference engine)
2) A knowledge base, not a knowledge unit, is persistent
3) An inference engine loads a knowledge base into its workspace - these are two different classes of objects. For example, a knowledge base inherits methods for editing the knowledge units, whereas an inference engine has methods (assert, retract, modify, ..) for modifying its workspace.

Class

TStatement

Description

A basic world (or solution state) description item. It is derived from a database or an event (a "fact"), or may be an artifact of the problem solving (a " hypothesis"). Incompletely instantiated statements are "patterns" or "constraints": pattern-directed inferencing is the basis of layer 4 execution.

Superclass

TKnowledgeUnit

Subclasses

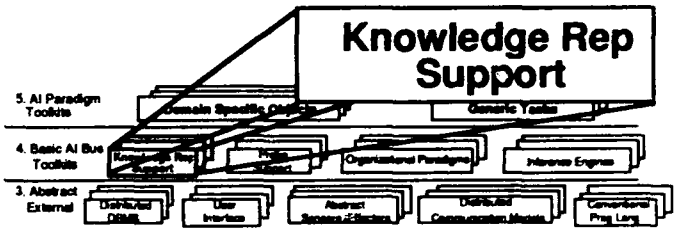
TUnnormalizedRelation, TFirstNFRelation

Policy

1) Statements are unified and tested by methods contained in an inference engine
2) Statements are considered to be declarative; attaching probes to a statement enables procedural capabilities found in some frame-based languages, such as querying the user ("if-needed" facets), and active value ("if added .." facets). That kind of embedded control is the responsibility of probes and not the inference engine.

Class

TRule



Description

A unit of inference: if conditions are true, then do actions and then draw conclusions

Superclass

TKnowledgeUnit

Subclasses

TClipsRule

Components

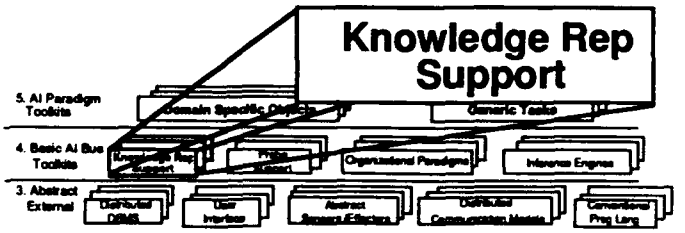
Priority
Trigger
Set of Actions
Set of Consequences

Policy

1) The operations on a rule (test, select, fire) are implemented in the inference engine
2) A rule does not have a "direction", that is a component of the inference engine
3) The priority is used by the inference engine's scheduler
4) The trigger is a join of statements, perhaps together with procedural tests, and perhaps a measure of uncertainty (and a threshold)
5) An action sends a message (to a user, an effector, a database or another knowledge source); that is, it has side-effects.
6) A consequence does not have side-effects, it specifies a modification to the inference engine's workspace

Class

TKnowledgeSource



Description

The basic homogeneous item of intelligence, a knowledge base and an inference engine.

Superclass

TSaveableObject

Components

TKnowledgeBase
TInferenceEngine
StateFlag (Active|Inactive)
Priority

Functions

Initialize : -> nil
Action: Loads the knowledgebase and initializes (resets) the inference engine

Run : NumInferences|Duration|UntilCondition -> nil
Action: Runs the inference engine for a number of inferences, or a time duration, or until some condition (e.g., no more inferences possible) is true

Suspend : -> nil
Action: Suspend execution of inference engine, set state flag to Inactive

Resume : -> nil
Action: Resume execution of inference engine, set state flag to Active

Assert : TKnowledgeUnit -> nil
Action: Asserts knowledge unit into inference engine's work space

---

Policy

1) Homogeneous knowledge representation: mixing knowledge representation schemes occurs at the agent level via an organizational paradigm

2) Running a knowledge source changes itself. The changes may be to the KnowledgeBase (learning facts & rules), or the InferenceEngine (e.g., changing OPS83 's RAC on the fly, or BB1's dynamic control blackboard).

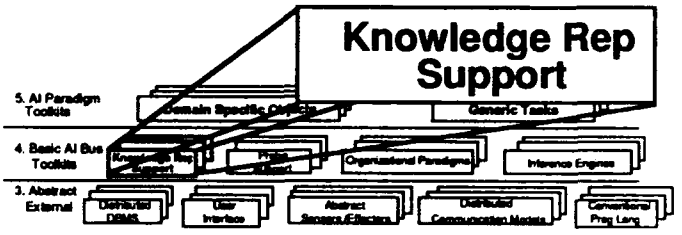
3) The priority may be used by an organizational paradigm's scheduler for combining several agendas of knowledge source activations (i.e. for global conflict resolution)

4) The assert will coerce the knowledge unit into the right type through the knowledge unit's coercion operator

5) For remote access, these methods will be called from an attached message manager

Class

TUnnormalizedRelation



Description

A place-holder for logic and OOP representations of statements, never instantiated; corresponds to logic and object-oriented databases

Superclass

TStatement

Subclasses

TLogicTerm, TFrame

Components

Functions

Class

TFirstNFRelation

Description

A place-holder for OPSxx representations of statements, never instantiated; corresponds to relational databases

Superclass

TStatement

Subclasses

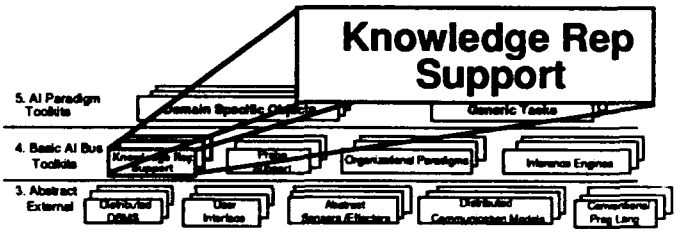
TClipsFact

Components

Functions

Class

TLogicTerm



Description

A predicate logic statement, such as Prolog or MRS; their structure has arbitrary depth, which necessitates recursive unification

Superclass

TUnnormalizedRelation

Components

Functions

Class

TFrame

Description

An object consisting of attributes and values, with inheritance and defaults

Superclass

TUnnormalizedRelation

Components

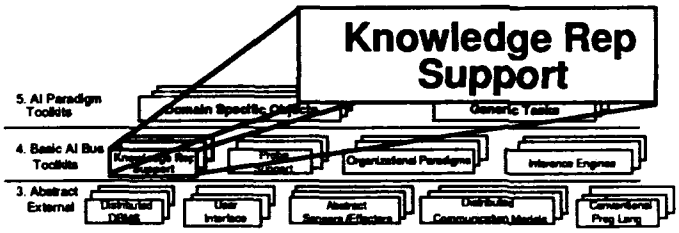
IsaSlot (inheritance info)
Set of slots, each of which is a set of facets

Functions

AddFrameData : Slot x Facet x value -> nil
Action: Stores and keys a value into a new facet
RetrieveFrameData : Slot x Facet -> value
Action: Accesses a keyed item in a facet

Class

TClipsFact



Description

A list of fields; a logic statement of depth 1

Superclass

TFirstNFRelation

Components

List of fields

Functions

Class

TClipsRule

Description

A rule interpreted by the Clips inference engine

Superclass

TRule

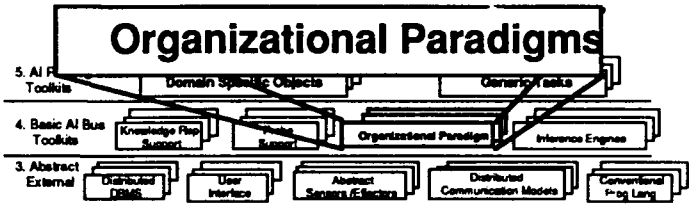
Components

Functions

4.3.2.4.2 Organizational Paradigm Specification

Class

TKSMessageManager



Description

A message to method call transducer object for attaching to Knowledge Source objects.

Superclass

TMessageManager

Components

Functions

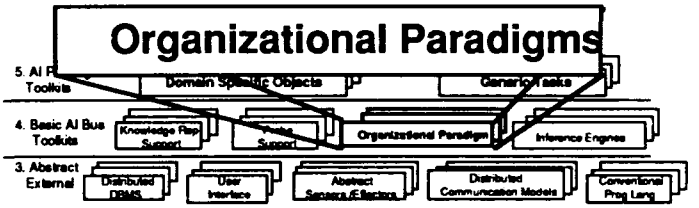
Policy

1) It adds knowledge to the associated knowledge source by invoking the KS's assert method

2) To avoid infinite loops, its Send method, which will be invoked from the KS's Run, should not itself invoke the KS's Run.

Class

TBBMessageManager



Description

A message to method call transducer object for attaching to Blackboard objects.

Superclass

TMessageManager

Components

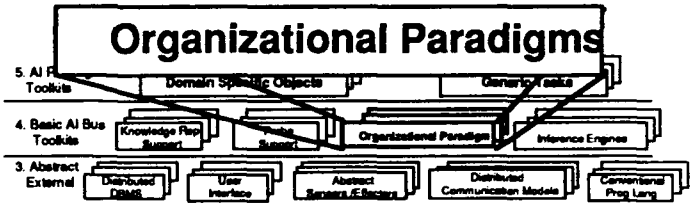
Functions

Policy

1) May contain some of Ensor's transactional manager capability

Class

TAgentMessageManager



Description

A message to method call transducer object for attaching to Agent objects. Preprocesses Ask and Tell requests for the agent by mapping to internal handlers, such as component Knowledge Sources, using the tables of capabilities and interests. Conversely, maps requests and notifications from the agent to external handlers (other agents, not known to the agent by name) using the table of acquaintances - this enables direct anonymous communication between agents while blackboards enable broadcast anonymous communication. See Fig. 4.11.

Superclass

TMessageManager

Components

Capability Table (Questions agent can be asked)
Interest Table (InfoItems it should be told)
Acquaintance Table (List of other Agents & their Capabilities & Interests)

Functions

ConstructCapabilities : Set of <message,objectID> pairs ->
Action: Called from attached agent to fill in capability table
ConstructInterests : Set of <message,objectID> pairs ->
Action: Called from attached agent to fill in interest table
Advertise : -> nil | Pair of message sets
Action: Returns capability messages and interest messages

Policy

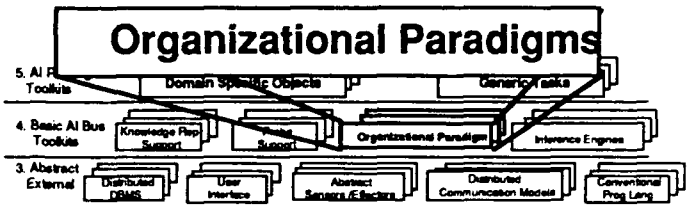
1) The Capability table has the form: < <*AskMessage InternalHandler*> .. >
The Interest table has the form: < <*TellMessage InternalHandler*> .. >
The Acquaintances has the form: < <*AskMessage AgentID*> .. > < <*TellMessage AgentID*> .. >
where the message can contain knowledge units with wildcards, etc. for intelligent pattern matching

2) Advertise returns nil if the capabilities and interests have not yet been constructed - the caller should try again later

3) The send message (called from attached agent) will be routed appropriately, either to an acquaintance, a blackboard, or to an agent directly if attached agent knows his name (i.e. he included the destination as part of the message himself).

Class

TBlackboard



Description

A broadcast semi-permanent communication space; attached probes provide for pattern-directed invocation of knowledge sources

Superclass

TProbeableObject

Components

WorkSpace (maybe with Hierarchical Structuring scheme)
Agenda (maintained by scheduler)
Control Loop
Transactional Manager

Functions

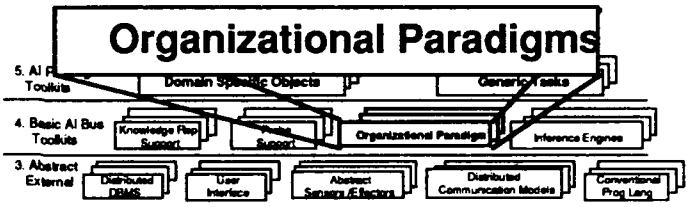
Restructure: Structure -> nil
Action: Dynamically modify structure of workspace
Add: Hyp -> nil
Action: Add hypothesis to workspace
Delete: Hyp -> nil
Action: Delete hypothesis from workspace
Retrieve: Trigger -> Hyp
Action: Retrieves matching hypothesis from workspace

Policy

1) A "blackboard" becomes a control mechanism when probes, representing the interests of knowledge sources, are attached.
2) The transactional manager actually interprets the hypothesis updates and writes them onto the work space.
3) For remote access, these methods will be called from an attached message manager

Class

TDataflow



Description

An asynchronous message-passing functional approach.

Superclass

TObject

Components

Set of I/O message queues

Functions

Policy

Class

TProcedure

Description

A synchronous procedural sequence of invocations

Superclass

TObject

Components

Set of I/O message queues
Sequence of invocations

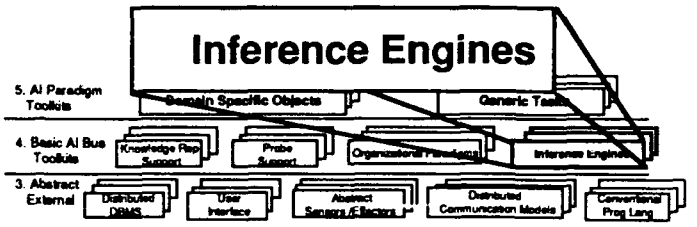
Functions

Policy

4.3.2.4.3 Inference Engine Specification

Class

TPatternMatch



Description

Provides unification between two statements.

Superclass

TObject

Components

Functions

Unify : Statement x Statement -> [Statement x Statement]|nil x BindingList

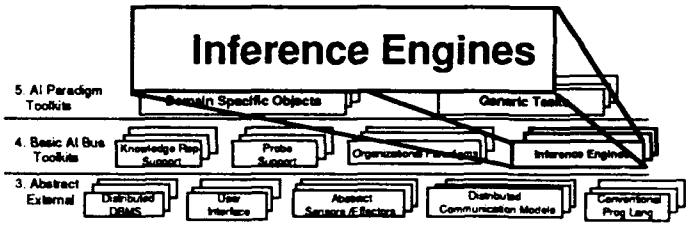
Action: Attempts to match two statements, returning them with variables bound as much as possible. If no match, return nil.

Policy

1) Invoked iteratively, is used for trigger invocation. As such, provides support for inference engines and probes.

Class

TTruthMaintenance



Description

Maintain dependencies for reasons of non-monotonic reasoning and explanation

Superclass

TObject

Components

Dependency Graph

Functions

Switch : On|Off -> nil

Action: Turns on/off dependency-maintenance

StoreDependency : TStatement -> nil

Action: Adds a dependency to the graph when a statement is asserted

DeleteDependency : TStatement -> nil

Action: Deletes a dependency from the graph when a statement is deleted

DetectContradiction : -> nil

Action: May invoke re-inferencing when a contradiction is detected

MaintainConsistency : -> nil

Action: Invokes re-inferencing if called when a contradiction is detected

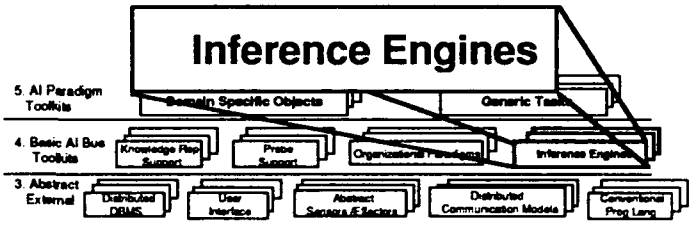
---

Policy

1) Used in explanation and "what-if" exploration

Class

TInferenceEngine



Description

A run-time environment for interpreting and manipulating a knowledge base.

Superclass

TSaveableObject

Components

TPatternmatch
TTruthMaintenance
ControlStrategy
WorkSpace
Agenda (with scheduler)

Functions

Load : TKnowledgeBase -> nil
Action: Constructs the Workspace from a knowledge base
Initialize : TMSswitch-> nil
Action: Initializes the agenda and other internal data structures, invoking truth maintenance system or not
Run : NumFires|Duration|UntilCondition-> KnowledgeBase
Action: Executes the inference engine
Explain : Statement -> Dependencies
Action: Outputs the history of reasoning (e.g., rule firings) that supports a statement
Suspend : -> nil
Action: Suspend execution of inference engine
Resume : -> nil
Action: Resume execution of inference engine
SaveState : -> TKnowledgeBase
Action: Create a knowledge base out of the current work space
Assert : TKnowledgeUnit -> nil
Action: Asserts knowledge unit into inference engine's work space

---

Policy

1) The ControlStrategy includes the direction of inference (forward or backward) and search strategy (depth-first, etc.)
2) Only one knowledge base can be stored in the workspace at a given time: the state of the workspace before the first load or after a second load is an issue for the trade study. Similarly, the sequence of requests should be load, initialize and run; a different order is incorrect but may default to some action.
3) Additional services may be required for uncertainty propagation and running mode (e.g., trace, dribble)

4.3.2.4.4 Probe Support

Probes are the basis for access oriented programming in the K-Bus architecture. Since they have potential performance impact, their implementation will be imbedded in lower layers. Probe support is represented by both objects at the layer 4 level, and methods that are a part of other objects at various levels. Probes' subclasses will differ in terms of the type of event that they monitor, and therefore the type of object to which they are attached. Objects which provide probe attachments will include a AttachProbe method that will link a probe to the object in that object's specific way, and a DetachProbe method, to undo the attachment. Objects that can have attached probes must also provide an attachedProbes method that delivers a collection of probes to the caller. There are some optional methods these attachable objects may also override.

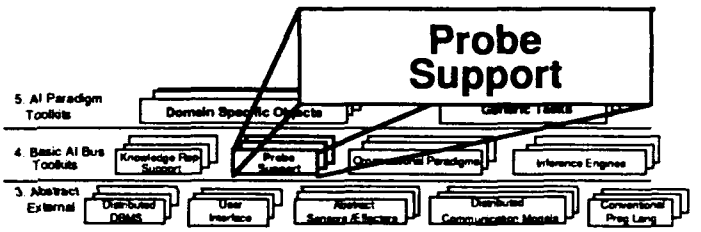
Currently envisioned objects to which probes can be attached include:

- DBMS objects allowing probes to trigger on events of READ, UPDATE, DELETE, and INSERT qualified by relation name, and values within the relation. Many applications can use this as their primary execution control mechanism and input data source.
- Data Communications objects that allow probes to trigger on particular message types qualified by sender or receiver. This allows applications to examine the data used by other applications, as well as giving applications an introspection capability.
- Blackboard Objects, allowing triggering based on operations similar to DBMS objects. The blackboard triggering combined with the its information structuring capabilities are what differentiate blackboards from shared memory.
- Agents, where triggering in terms of agent state as can be determined by one of the agent knowledge sources (demon rule)

The probe mechanism provides for easy attachment of new facilities to an existing system built on the K-Bus. The new facilities would use probes to access their needed data, without requiring changes to other programs that use the data. Supervisory and monitoring applications are especially appropriate to this form of interface to a system. Probes also provide a convenient test and validation view into an operating distributed application suite.

Class

TProbe



Description

The demon object. Contains the requirements for the pattern directed firing, and the method to be carried out (fire) when the pattern is matched.

Superclass

TSaveableObject

Subclasses

Components

Priority
EventDescriptor
Trigger
EventHistory
TPostOffice reference
routingList

Functions

Fire : eventObject -> nil

Action: Executes the probes action. Generally this action will involve communicating some information to its owner /supervisor object.

SetPO : TPostOffice ->

Action: Puts the postOffice reference in the probe, providing a vehicle for corresponding with the outside world.

Policy

1) The object to which the probe is attached implements the trigger mechanism based on the rule's trigger (i.e., the send method). The rule trigger must meet the particular requirements of the object type.

2) The eventObject passed when a TProbe rule is to be fired is related to the object to which the probe is attached. For database objects, for instance, it is the tuple involved in the database access event.
3) The action component of the probe is implemented via a method for the object. The default Fire method does nothing, and is generally overridden for a particular probe.
4) The priority is established at probe creation time, and is used to handle conflict resolution when multiple probes are active
5) There is a TProbeableObject Class that provides default methods for objects that can have probes attached. All objects that may have probes attached will be a subclass of this object class .
6) An attached probe exists in the process space of the object to which it is attached and may be remote to its "supervisor". On attaching, the probeable object is required to supply a PostOffice for the probes use through a SetPO call.
7) The routingList provides a keyword list of object identifiers used in the probe's Fire method to pass information from its monitored object to the probe's external audience.

4.3.2.5 AI Paradigm Toolkit Layer Design Specification

The Layer 4 objects - knowledge sources, probes and organizational models - are combined into the fundamental module of intelligence, an agent. Figure 4.10 illustrates the structure of an agent composed of two knowledge sources which communicate (through their message managers) indirectly via a blackboard. Note that this is simply an example, and it is possible to have knowledge sources communicating directly with each other or via some other other organization.

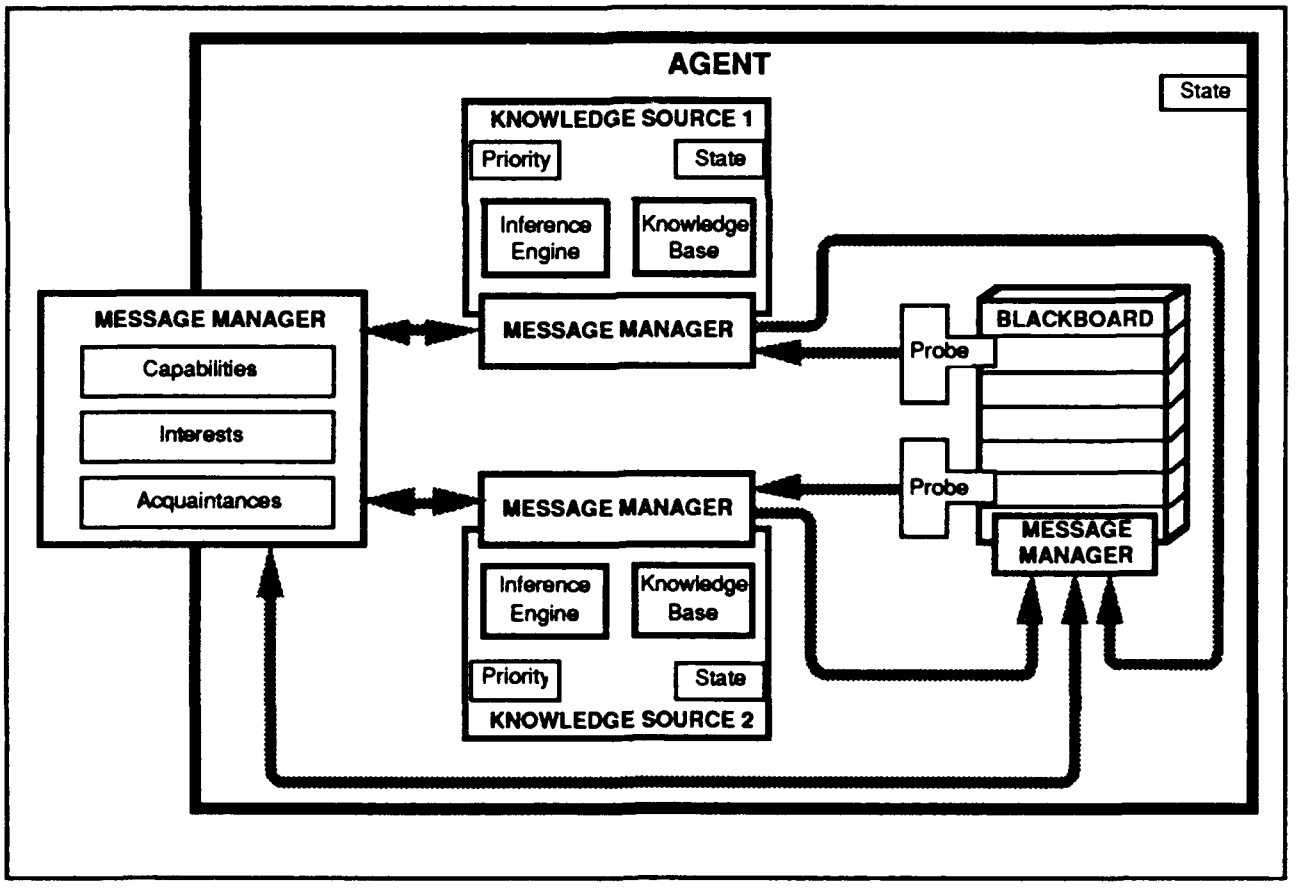


Figure 4.10. An Example of an Agent Composed of Several Layer 4 Objects

Agents advertise their capabilities and interests to other agents via their message managers, allowing them to construct models of their acquaintances - which agent(s) to ask for information and which agent(s) are interested in

being told information. Alternatively, indirect communication is possible between agents using a blackboard, for example. These possibilities are illustrated in Figure 4.11.

Class

TAgent

Description

A heterogeneous single-task semi-autonomous expert.

Superclass

TSaveableObject, TProbeableObject

Components

Set of KnowledgeSources
Set of Probes
Organizational Paradigm Objects
Active|Inactive StateFlag

Functions

Install : nil -> nil
Action: Loads the agent's description into image form, initializing its components
Remove : nil -> nil
Action: Removes itself from memory, killing all processes
Ask : Message -> Message
Action: Passes the message to the agent and returns the answer. The caller may or may not block until it gets the answer.
Tell : Message ->
Action: Passes the message to the agent

Policy

1) For remote access, these methods will be called from the attached message manager, which acts as an internal and external interface. Thus, it maps the capabilities and interests to the agent's knowledge sources which can field the requests. Also, it maps external requests to other agents according to their capabilities and interests.
2) Inter-agent communication can be goal-driven (Ask) or event-driven (Tell) but not control-driven as they can't see each other's internals and so can't invoke private actions

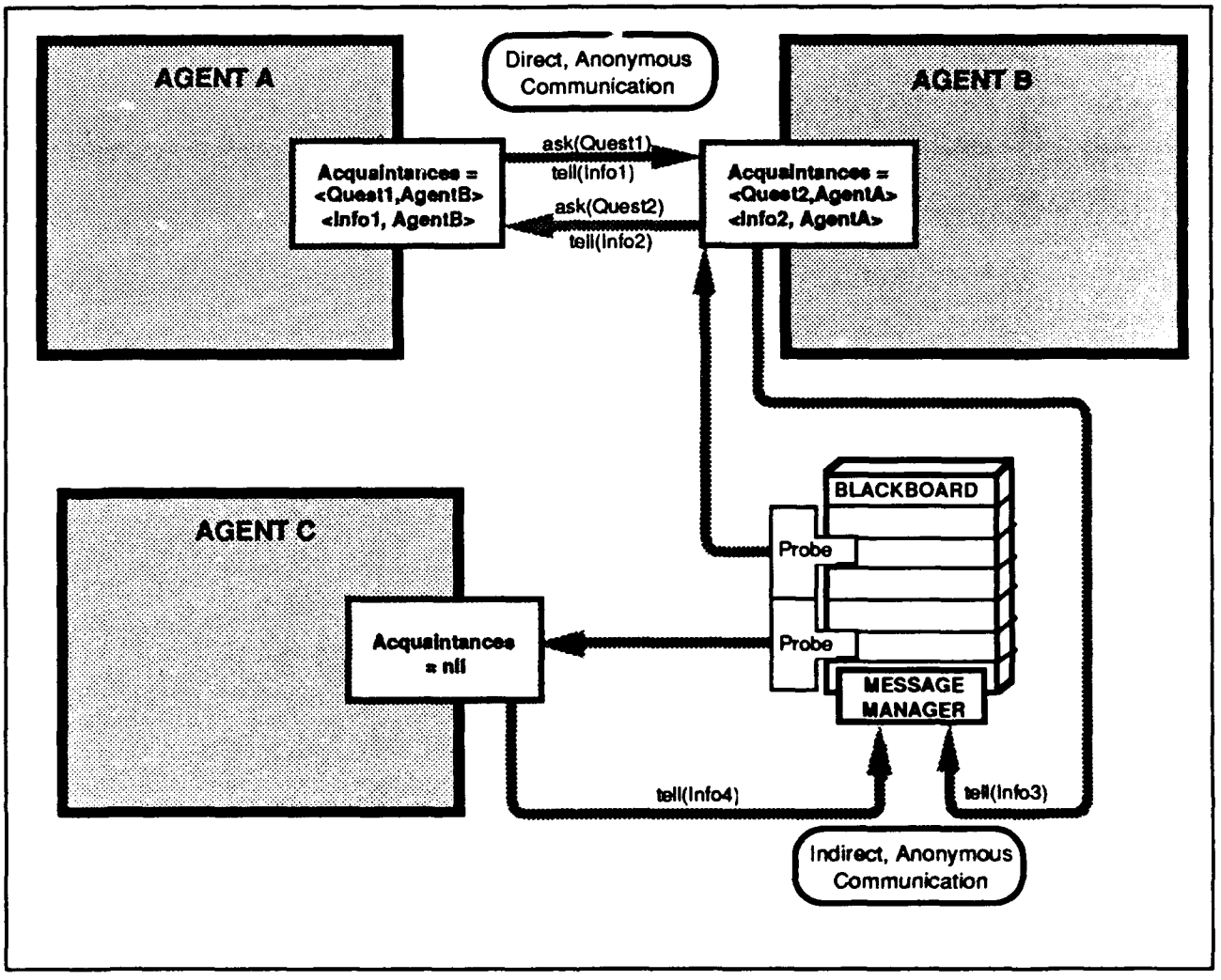


Figure 4.11. Agents Communicate directly with their Acquaintances and indirectly via Blackboards

Class

TAgent

Description

A heterogeneous single-task semi-autonomous expert.

Superclass

TSaveableObject, TProbeableObject

Components

Set of KnowledgeSources
Set of Probes
Organizational Paradigm Objects
Active|Inactive StateFlag

Functions

Install : nil -> nil
Action: Loads the agent's description into image form, initializing its components
Remove : nil -> nil
Action: Removes itself from memory, killing all processes
Ask : Message -> Message
Action: Passes the message to the agent and returns the answer. The caller may or may not block until it gets the answer.
Tell : Message ->
Action: Passes the message to the agent

Policy

1) For remote access, these methods will be called from the attached message manager, which acts as an internal and external interface. Thus, it maps the capabilities and interests to the agent's knowledge sources which can field the requests. Also, it maps external requests to other agents according to their capabilities and interests.
2) Inter-agent communication can be goal-driven (Ask) or event-driven (Tell) but not control-driven as they can't see each other's internals and so can't invoke private actions

4.3.2.6 Generic Application Layer Design Specification

No generic applications were identified as being sufficiently well defined to merit a class specification at this time. This layer is a placeholder for future research advances.

4.3.2.7 Application Layer Design Specification

Class

TApplication

Description

An executable program, a collection of interacting agents

Superclass

TSaveableObject

Components

Set of Agents (including at least a bootstrap agent)
Organizational Paradigm
Set of AuditProbes

Functions

Start : Command -> Application

Action: Invokes its bootstrap agent(s) to install the other agents, according to the command line parameters

Stop : nil -> nil

Action: Removes all agents in the application

Policy

1) The overall VMMS would be a layer 7 application

4.4 K-BUS TRADE STUDIES

4.4.1 Trade Studies Overview

The multiple goals of the K-Bus architecture often pull design decisions in multiple directions, where facets of some goals are compromised to better meet others. The design specification of 4.3 represents a preliminary design, and as such is subject to revisions as the design is elaborated further and is tested through prototyping. This section provides access to current design rationale, and to current recognized deficiencies and open issues. As such, it serves as a working agenda of items to be addressed in further elaboration of the specification.

This section is organized based on the K-Bus layers. Some issues transcend layers; in these cases, we've attempted to place their discussion in the layer one would most associate with the issue.

4.4.2 Layer Trades

4.4.2.1 Devices & Users Layer

None identified.

4.4.2.1.1 Devices & Users Layer Design Issues & Trades

4.4.2.1.2 Devices & Users Layer Implementation Issues & Trades

4.4.2.2 Concrete External World Layer

The primary issues related to layer 2, the concrete external world layer, revolve around the particular abstractions needed to provide adequate access to the layer 2 concrete world services, yet provide high portability to components using abstractions in this layer.

Although not a real-time operating system, UNIX provides a fairly complete and easy to use operating system interface that could be used as the interface abstraction even when the underlying system is not UNIX. This approach was chosen because any other involved either choosing a different operating system as the interface

paradigm or inventing a new system interface, which would be untried. UNIX provided the advantage that it is the underlying system for the AI testbed, it has been shown to be effective on many different hardware platforms, and it has an open and well-documented structure. Since it is not real-time, we anticipate adding to the interface abstraction as needs for real-time specific services are required, as driven by the anticipated applications.

Other issues involve device and network interfaces, and the mappings from logical to physical. These are detailed in the tables that follow, which summarize the issues and the current approach being taken.

4.4.2.2.1 Concrete External World Layer Design Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Operating System Interface	TSystemCall	Create a single object interface for all OS calls, or create a suite of objects that better map OS functionality	Single Object	Less conceptual clutter, easier to handle call interaction.
Device Interfaces		Are special device interfaces needed at level 2, or should the abstract sensor and effector objects handle the operating system isolation.	Object very similar to UNIX device driver	UNIX device driver abstraction simple character stream
Socket Abstraction	TSocket	The sockets are an abstraction of the BSD 4.3 sockets, although implementation may not necessarily use sockets. Is there some more abstract approach that better fits the Object Oriented design philosophy?	Socket abstraction	Highly general, chosen by Apollo for NCS underlying abstraction
Socket Addressing	TSocket	How is mapping from logical to physical socket names to occur? Is just one name server adequate, or are two levels of addressing needed.	TBD	

4.4.2.2.2 Concrete External World Layer Implementation Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
TSocket Implementation	TSocket	Are sockets too inefficient to use as an implementation for the socket abstraction?	Assume BSD sockets can be used. Investigate other approaches when on same machine.	
Socket Addressing	TSocket	If using UNIX BSD sockets as an implementation of the socket abstraction, where is the mapping to physical addressing to occur?	TBD	

Buffers	TDevice	Should buffers be objects?	Unstructured	Efficiency
Streams	TDevices	Do streams provide a more powerful abstraction for device interfacing?	Under investigation	Need to get further streams documentation

4.4.2.3 Abstract External World Layer

The issues in the abstract external world layer concern identification of the optimal set of underlying abstractions of traditional distributed data processing functions to meet the needs of the higher abstraction layers. The goal is to provide program independence from underlying DBMSs, User interface management systems (UIMSs), device and data streams, and the physical distribution of application system components.

4.4.2.3.1 Abstract External World Layer Design Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Messages	TSocket	Need a message standard, or at least a standard message envelope? Probes may need certain buried information.	Develop a standard message object class	Abstraction above implementation guarantees certain message info will be available.
Probes & databases	TDBMS, TProbe	Databases will probably be a popular object of probes. How can this be done efficiently?	TDBMS implements an efficient trigger. Some underlying DBMSs may provide a trigger mechanism.	A generic trigger for probes appears impossible due to the differences between probed objects.
Database access objects	TDBAccess	How object oriented can a traditional DBMS be made too look? Should the normal query interface be still available?	Provide as a minimum, an objectized query interface, then use this to create a more object oriented DBMS service	Traditional queries needed for hosting off-the-shelf software & for efficiency reasons.
Higher order DBMS abstractions	TDBFetchPipe TDBPutPipe	Higher order objects are needed to handle persistent objects of a more abstract nature.	TSaveableObject class objects know how to store themselves	Separates traditional & object oriented DBMS concerns
Query as Object	TDBQuery	Does this need to be an object, or just a structure?	If structures are regarded as on form of class, this is a moot point	Choice of C++ as an object language model has this view
Dealing with flat files	TDBMS, TDBAccess	Do we need special objects to deal with flat files?	Use DBMS objects	Appear to be a good fit. May need to provide simple SQL parsing for flat files.

Records in flat files	TDBMS, TDBAccess	Should records be parsed and split into separate field objects by the flat file handling part of the database objects?	Keep record as single field	Existing programs accessing files do their own parsing, new programs can simply use the DBMS.
User Interface	TScreen, TPanel	Current abstraction for the UIMS is based on TAE. There may be a more general abstraction, that can be implemented with TAE, which provides more flexibility, and matches the Object Oriented paradigm better	TBD	Examining InterViews, Open Look, and other X-Window based toolkits for applicable ideas
TVm UI Object	TVm	The TAE TVm object has a lot in common with the TItemList object used for database access. Should they be a single object type?	TBD	Need to examine the latest TAE release to see if changes.
Scaling Support for sensors	TSensor	Is there a way to provide reusable support for scaling operations. Also averaging & interpolation services.	TBD	Need more information on sensor types envisioned

4.4.2.3.2 Abstract External World Layer Implementation Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Attaching abstract sensors & effectors	TSensor, TEffector	How will TSensor & Effector objects be tied to related TDriver. Need ability to switch underlying data sources transparent to clients.	Follow UNIX logical file access approach	UNIX provides this source data independence if clients follow protocol
Pre & post processing of sensor readings	TSensor, TEffector	Need reusable sensor pre and post processing when possible. The UNIX stream stack approach may be a good paradigm to follow in designing the TSensor & TEffector object classes.	TBD	Look into creation of tools to generate stream stacks. See sensor& effector coding below.
Sensor & Effector coding	TSensor, TEffector	Can we make creation of abstract sensor and effector code easier	Envision toolkits, possibly using direct manipulation UI	Rapid prototyping, rapid reconfiguration

Sensor/Effector response	TSensor, TEffector	Sensors and effectors may need much higher data rates than K-Bus objects can supply. How is the bandwidth gap bridged? How to deal with sensor/effector control loops with tight response time requirements.	More intelligence in the abstract sensor effectors. Slower K-Bus components continuously monitor and reconfigure higher data rate control loops that may be resident in sensor/effectors own processor.	Sensor/Effector toolkits may need to support closed control loops.
Name Server	TFinder	Can existing UNIX name servers provide the TFinder service, or do we need something beyond this? How are various name servers informed of changes?	Treat similar to distributed DBMS, but use memory cache for performance.	Phase II TBD.
Location Transparency	TMessage-Manager	This object provides remote access to certain objects, with whom it shares a process, by translating messages (events) to method calls. A source object either invokes its message manager to "send" to a target object if it's remote, or calls that object directly if its local. A desire for transparency might hide this location distinction from the source programmer under a common syntax, by requiring every object to have a message manager, which may be of type remote or local (two subclasses), as in NCS stubs.	Only one class: the use of the message manager is specified as being for location transparent access to objects. Objects communicated with this way must also have message managers. The message manager can communicate with message managers that are local or remote.	Definition may change later after implementation experience, at the moment it is the programmer's responsibility to use the appropriate tools correctly

4.4.2.4 Basic K-Bus Toolkit Layer

4.4.2.4.1 Basic K-Bus Toolkit Layer Design Issues & Trades

Title	Objects	Issue	Current Approach	Rationale

Knowledge source suspension	TKnowledge-Source	The Suspend method may need a parameter which determines what should happen to asserts while the knowledge source is suspended. The message manager will continue receiving message; they could be ignored or still be transferred into the inference engine's workspace. The latter option would be chosen if there is a concern that the queue would become full and information would be lost.	No parameter in method. The implementation assumes that the inputs are ignored while the knowledge source is suspended.	Easily added later if necessary, but see implementation issues section for reason why the other option may be hard to implement.
Blackboard Retrieve	TBlackboard	Retrieve method may be unnecessary if probes are always attached for this purpose	Method exists	Caution - keep in case needed
Probe context	TProbe	How are the references in the trigger & action linked to data in the environment the probe is to be installed within? It seems the probe lives in both the address space of its host object, and the address space of its owner object. We could restrict the host object addressing by the probe to the eventObject; does this limit us very much?	Host object is required to deliver certain event oriented information. This is the total context (other than its own memory) for the probes code.	Efficiency and flexibility.
Probe capabilities	TProbe	Should a probe be an agent, with all the attendant smarts, or should a probe be more like an abstract sensor, and simply access and deliver data, to be interpreted by its client?	Probe more like sensor	Efficiency and more obvious implementation.
Probe History	TProbe	Probes as originally envisioned were provided with an event history. Should this history (if provided) be a history of all events of the type the probe is monitoring, or only those that match the probes trigger pattern.	TBD	
Probe Communication	TProbe	How do probes communicate with their "owners"?	Through the TPostOffice of their TProbeableObject, who is responsible for providing a reference to his PostOffice on attachment of the probe	Good fit with envisioned communication models

4.4.2.4.2 Basic K-Bus Toolkit Layer Implementation Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Critical Section for Inference Engine	TInference-Engine	A knowledge source may be suspended, and information must be asserted into the inference engine's workspace. Both events require interrupting the inference engine's recognize-act cycle. Where is this safe? A few systems allow the RAC to be interrupted, most only allow external control at the end of a cycle. This is an issue in real-time control, where the prompt response to alarms and interrupts is critical.	This is not specified in the design, but we expect: the RAC cannot be interrupted; all RHS actions of a fired rule must complete before an external event is handled. The input queue is polled by the inference engine every cycle.	The semantics of a program whose facts are changing within the interpretation of a rule is considered too difficult to understand by programmers.
Granularity of inference engine components	TInference-Engine	Some separation of inference engine components is made (pattern match, truth maintenance), but there could be more extensive decomposition into lower level services. It is a question of whether there is a need to interchange these components (say, for efficiency flexibility, or reusability).	Some components are identified as objects in their own right.	Some flexibility is expected, however most programming is at a higher level and no interface to finer-grain services is anticipated in the near term.
Probe pattern match support	TProbeable-Object	There may be commonality in the pattern match required for attached probes. This might be added as an internal method of TProbeableObject	TBD	

4.4.2.5 AI Paradigm Toolkit Layer

4.4.2.5.1 AI Paradigm Toolkit Layer Design Issues & Trades

Title	Objects	Issue	Current Approach	Rationale

Agent definition	TAgent	The bottom up categorization of an agent could be recursive - an agent could be composed of other agents. In that case there would be no need for a knowledge source object (it would be an agent), and the design would be simpler and more uniform.	We differentiate between a knowledge source which is a module with one inference engine (as with most traditional KBS's), and agent which can have several.	This is just one way to categorize modules, and may later be later replaced with another.
Agent creation	TAgent	How do agents create other agents? There may be a need for a Create method, prior to a call to the new agent's Install method. This may be a bootstrapping concern - how do persistent versions of agents get instantiated into image form?	TCloner class object will be assumed to start automatically on each platform. This object class is intended to provide remote instantiation of active objects. One could implement to include a initial start-up agenda, so that some agents were automatically started at boot time.	TCloner may be sufficient, implementation experience is needed.

4.4.2.5.2 AI Paradigm Toolkit Layer Implementation Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Physical mapping of agent to process	TAgent	Agents are considered to be in control of a logical process, shared with their message manager. This logical process may be composed of several concurrent physical processes, for example one per knowledge source, but on the other hand, two agents may share a physical process.	Not specified. The definition of an active object has yet to be made, as has the mapping of physical distribution to logical. Message managers were introduced to deal with this question in an explicit way.	This allows flexible implementation, experience may help tie this issue down.
Inter-agent Communication	TAgent	The TAgentMessageManager is used as a front-end processor for routing internal and external requests. Its cache of tables of capabilities, interests and acquaintances could reside as components of the agent itself.	Message manager contains the tables	Logically keeps routing and query interpretation in one place, and physically localizes tables instead of being concerned about where they should reside if agent is spread across several processes.

4.4.2.6 Generic Application Layer

4.4.2.6.1 Generic Application Layer Design Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Identify Objects	TBD	No generic applications were identified in sufficient detail to be specified. This may be dependent on identifying generic tasks at layer 5, but on the other hand a generic application may be composed directly of lower level tools.	More implement-ation experience is needed to extract common denominators from applications built using the K-Bus.	TBD

4.4.2.6.2 Generic Application Layer Implementation Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
High-level language	TBD	A generic application may need more than an class definition, it may need to be implemented as a whole new language. The library of classes supplies the nouns and verbs, but not the grammar (i.e the usage) - but a generic application may need strictly enforced usage rules.	TBD	TBD

4.4.2.7 Application Layer

4.4.2.7.1 Application Layer Design Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Command Line arguments	TApplication	The input parameter ("Command") to Start needs elaboration. Alternatively, more methods may be needed to cover the variety of requests made from an application. If generic applications are defined, these would be invoked with parameters specifying the particular application.	Unspecified command syntax and semantics.	Implementation experience needed, as well as definition of a generic application.

4.4.2.7.2 Application Layer Implementation Issues & Trades

Title	Objects	Issue	Current Approach	Rationale
Procedural Program	TApplication	Tracing down the implementation hierarchy, agents are defined as having knowledge sources as components. However this does not rule out procedural components (they simply aren't mentioned) or a purely procedural program (all sets mentioned may be null).	A purely procedural program would consist of agents which respond to Ask and Tell by executing conventional code.	Satisfactory

4.5 K-BUS PRELIMINARY DESIGN ASSESSMENT

4.5.1 Usage Scenario

4.5.1.1 Introduction

As a test case for the K-Bus concept, a practical application with elements representative of ALS requirements is discussed. The scenario is modeled on a program, EXIMU, which monitors inertial measurement units aboard the Space Shuttle, which was developed by ABACUS under subcontract for NASA, and has been used on every Shuttle mission since 1985.

This section has three following parts: a description of the problem of IMU monitoring, a possible design for EXIMU expressed in terms of objects and a possible mapping between these design objects and K-Bus objects. The benefits and risks of using the K-Bus in this usage scenario are discussed in section 4.5.2.

4.5.1.2 Problem Description

A Space Shuttle IMU (Inertial Measurement Unit) is a complex device which provides attitude information to the on-board general purpose computer (GPC) for inertial navigation. It consists of gyros, accelerometers, motors, heaters, A/D converters, BITE etc., together with software which compensates for known hardware errors (such as drift and bias) by controlling the hardware and applying corrections to the data transmitted to the GPC. The Space Shuttle has a cluster of three redundant IMUs which are tested and calibrated for 10 hours immediately prior to launch; stringent requirements must be satisfied as a precondition for liftoff. It is the purpose of EXIMU to aid the responsible engineers in their analysis of this complex, real-time test data. Because the IMUs' performance degrades over time, part of the analysis includes predicting whether future errors will be acceptable if the liftoff is delayed: if not acceptable, the IMUs will be calibrated again, resulting in a further delay. The two major objectives are:

- Safety - by identifying a faulty IMU
- Economy - by not scrubbing a launch because of a false alarm, and not holding a launch for an unnecessary recalibration.

4.5.1.3 Program Requirements

For the purposes of EXIMU, the IMU cluster itself can be considered to be a black box, whose outputs are monitored and analyzed according to formal and informal criteria. These outputs are communicated to EXIMU via a telemetry stream, which is preprocessed so that some formatting and scaling is done prior to reception. The inputs to the IMUs are not under the program's control, but are commanded by the GPC according to a feedback loop. Furthermore, IMUs cannot be repaired without first being replaced, so the emphasis is on fault detection, and diagnosis only as far as detecting easily corrected operational or communication errors. The formal criteria ("Launch Commit Criteria") are well documented, whereas the informal criteria consist of rules of thumb and algorithms used

by experienced engineers. Both sets of criteria involve limit checking (against historical measurements or between IMU pairs), real-time trend analysis, prediction of future state and pattern recognition against fault scenarios.

The monitoring takes place according to a predetermined timeline, and consists of several consecutive operational modes (e.g. attitude determination, preflight calibration, gyrocompass alignment), each of which has associated pass/fail criteria. These modes are identified by measurements in the telemetry stream. The nominal timeline may be altered dynamically by launch holds and recycling. Data is available but invalid outside its timeline interval.

The user-interface consists of windows displaying current IMU status, represented by measurement values displayed in tabular and graphical form, trends and analysis. Further windows display problem-solving capabilities such as diagnosis and explanation .

To summarize, the characteristics of the program are as follows:

- Real-time input data stream (10^2 Hz)
- Data stream contains errors (dropout, out-of-sync, noise) and imprecision
- Consecutive modes of operation
- Data validity depends on time (i.e. operational mode) - all data has an expiration time -- temporal reasoning and prediction of future status
- Historical database of measurements and error signatures
- Only data invalidity and user errors can be recovered from (no repair)
- Several windows on data
- "What-if" capabilities for manual overriding of inferencing
- Algorithmic calculations and heuristic inferencing
- Formal and informal rules of success/failure
- Simulation needed for development and testing

Although this application is just one component in a possible overall mission management system, there are several natural ways to partition the problem in order to take advantage of concurrent processing capabilities.

4.5.1.4 Design Objects Identified

There are five categories of objects: Data transformers, Database managers, Status notice boards, Intelligent agents, User-Interface managers, all of which may use services from a library of statistical and engineering algorithms.

The overall design of the application is illustrated in Figure 4.12. The user interface is left out for clarity.

The five categories of objects are described next.

1) Data Transformation

These objects are responsible for interfacing with the telemetry: receiving and pre-processing measurements, post-processing measurement requests and sending them to the IMU cluster. The objects are as follows:

IMU Cluster or Simulation

The IMU cluster can be considered as a command-driven stream source. Automatically transmitted measurements are: the time (once a second), the operational mode (when it changes), detected datacomm problems. Other measurements are sent according to commands. The commands accepted are:

1. **Initialize** : establish connection, start sending current values of automatic measurements
2. **Start <measurement list>** : send measurements' values once, and thereafter when they change (i.e. updates)

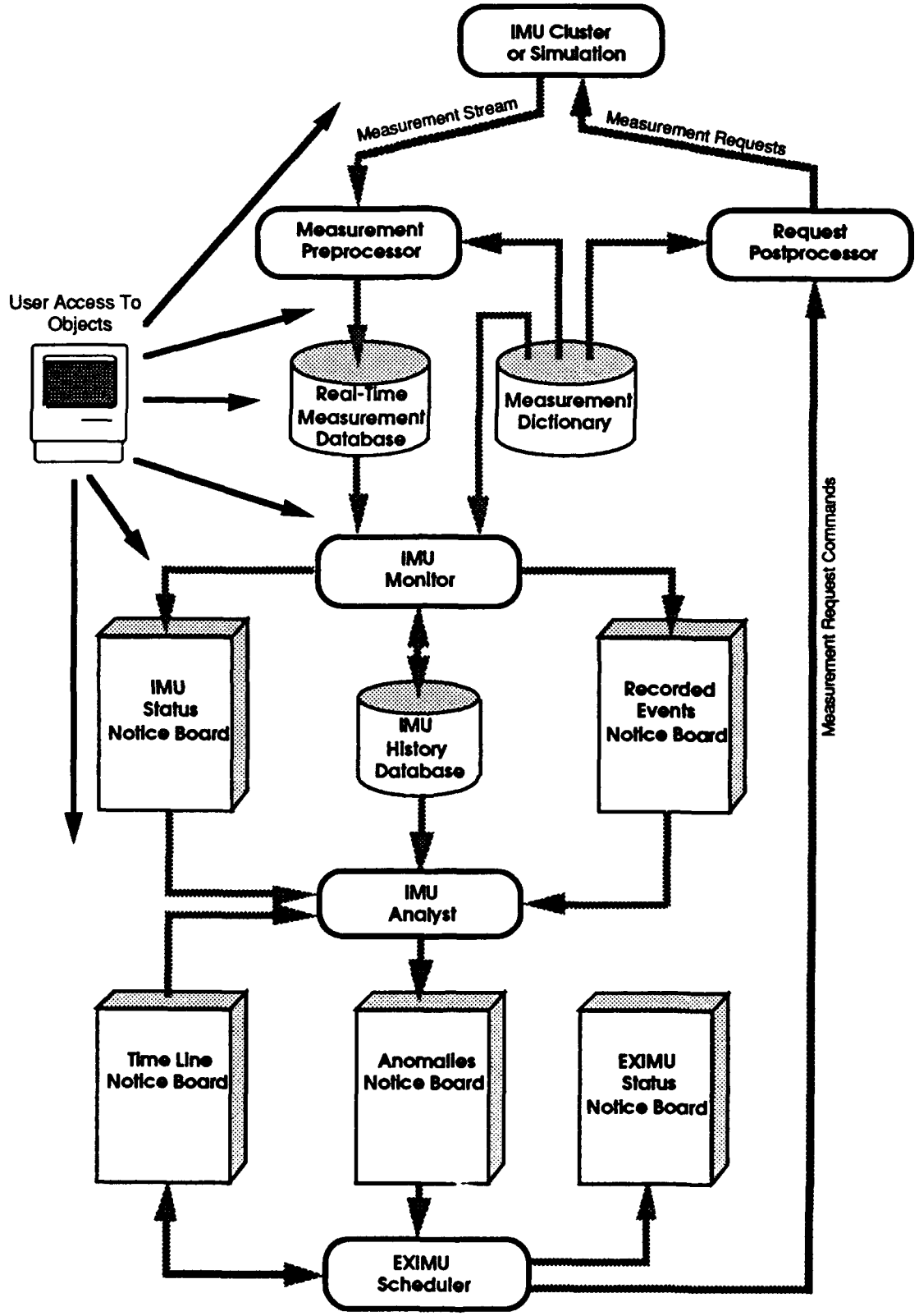


Figure 4.12. The EXIMU Application Design

3. **Stop <measurement list>** : stop sending measurements values

Simulation is achieved by implementing the telemetry source, with hooks for canned scenarios and interactive control.

Measurement Stream

Every measurement (including automatic) has a unique id. The stream consists of a sequence of items of the following form:

< <measurement id> *start* <text string> *stop* >

where start and stop delimit the measurement's value (transmitted as a text string). Note that the stream contains not all possible measurement updates, but only the subset specified in the start parameters.

Measurement Preprocessor

This translates the telemetry stream into a stream of measurement records of the following form:

< <measurement id> <time-tag> *pass/fail* <value> <imu> <subsystem> >

where pass/fail is a flag denoting whether the constraints were satisfied, and time-tag is the value of the most recent time transmission. This translation is performed by using information stored in the Measurement Dictionary database. If the data is recognized as invalid (because of datacomm problems or timeline mismatch) an alarm measurement should be sent and no further data communicated until it is again valid. Similarly, if filtering algorithms are specified for certain measurements, they will be used in the translation.

Request Postprocessor

This translates requests from the EXIMU scheduler agent into measurement requests, sent by telemetry to the IMU cluster.

2) Database Management

These objects store persistent data for rapid access and update.

Measurement Dictionary

This is a constant (i.e. access only) database which is used as for information about actual and derived (i.e. computed) measurements. It has records of the following form:

< <operation mode> <measurement id> <format> <constraints> <imu> <subsystem> >

where the format is real, integer, time, string, etc. and constraints are limits, time validity, etc. In addition, the formal rules, or Launch Commit Criteria (LCC), are stored here, and include both the constraints on actual measurements and also rules relating derived measurements This dictionary is used in translating raw measurements and requests, and by the IMU monitor agent.

Real-Time Measurement Database

This is a database of measurement records, generated by the measurement preprocessor. Its initialization conditions tell it how much data should be stored (e.g. 1 minute's worth) and what internal structure to use - measurements grouped together by IMU, subsystem, operational mode, etc. It accepts standard database queries, but it cannot be edited. It is not permanent and serves as a data source for the IMU Monitor agent.

IMU History Database

This permanent database contains historical data and filtered current data. The historical data comes from previous calibrations and tests of the IMUs, the current data is stored from the real-time database when certain specified conditions are met (e.g. an operational mode is complete). The historical data is identified according to type (e.g. calibrated in the factory, in the lab, in the shuttle). All data is tagged according to date and time. For efficiency

reasons, some derived data (e.g. statistical values, graphic plots) may also be stored if it's expensive to calculate dynamically. Included in the historical data are error signatures of the IMUs.

3) Status Notice Boards

These are indexed repositories for high-level event notifications and summaries of problem-solving status. They are used for communication between agents.

IMU Status Notice Board

This contains data (input, and derived from algorithms) relevant to the current operational mode. The data is generated by the IMU Monitor agent and monitored by the IMU Analyst agent.

Recorded Events Notice Board

This contains a log of transmitted and concluded events identified during the current program run. This will include summaries of previous operational modes' pass/fail conclusions (although if it failed, subsequent modes will be put on hold).

Time Line Notice Board

This contains the sequence of expected and actual event times and durations. Its initial state (i.e. the nominal time line) is determined by the mission plan, during the program run the actual times are written next to the expected times by the IMU Monitor agent. The expected times are used as context information by the IMU Analyst and EXIMU Scheduler agents to determine what operational mode is current. The expected times of future events are updated by the EXIMU Scheduler as a result of re-planning because of launch holds. These expected times are a summary of predictions made by EXIMU.

Anomalies Notice Board

This contains notifications of anomalies detected by the IMU Analyst. If the anomalies are not fatal, the information is used by the EXIMU Scheduler to re-plan the launch schedule.

EXIMU Status Notice Board

This summarizes the current status of EXIMU's problem solving. The form of this may be a description of what task is currently scheduled by the EXIMU scheduler, together with an explanation in terms of the current goals and plans that the tasks are designed to achieve.

4) Intelligent Agents

These modules direct the control of the application, based on evaluation of formal and informal rules.

IMU Monitor

This module is responsible for reading the real-time measurement records, calculating derived measurements and checking constraints, updating the history database with the current data, and posting its results to the IMU Status and Recorded Events notice boards. It is event-driven.

IMU Analyst

This module is responsible for analyzing the current data and making conclusions about the health of the IMUs. Its input comes from historical and current data, viewed in the context of the time line, and it is able to identify and diagnose faults with the IMUs, datacomm system, etc. If it diagnoses a real anomaly, then it posts this result to be used by the scheduler.

EXIMU Scheduler

This module is responsible for maintaining the high-level program plan (the time line events) and initiating actions (telemetry measurement requests) to realize that plan. If an anomaly is detected by the analyst, the scheduler modifies the plan by changing predictions of expected events on the Time Line notice board (because a launch hold will take place for another calibration, for example). The scheduler agent also accepts commands from the user which may override its own plan. It is goal-driven, its rules expressing constraints on events, priorities, times and actions.

5) User Interface Manager

This module is responsible for displaying sets of values and responding to requests from the user. It consists of windows, several of which can coexist on one screen, can take several forms (tabular and graphic) and which know how to reformat, scale, pan & zoom, resize and reposition themselves.

Every object identified above has an associated window attached which provides the user access to its inner workings. Interesting examples are as follows:

Real-time Window

This is a window on the real-time measurement database, corresponding quite closely to the capabilities of traditional monitoring display systems. It has a list of measurements to display, a description of how they should be displayed (location, color, etc.), what to display when a measurement's fail flag is set (e.g. redline, beep), how often to display the measurements (e.g. once a second). Special measurements, such as automatic ones, may be identified for special handling (e.g. alarms when mode changes or sync is lost). The values are continuously retrieved from the real-time database by standard DBMS queries, the form of which determine to an extent the structure of the real-time database. The user's input is restricted to calling up the window from a menu, responding to alarms, and closing the window. The display may show more than just the most recent values, for example a plot over (recent time) may be interesting.

Status Windows

These are windows on the Status Notice Boards. They provide a view of the current and past state of the IMUs, and the current state of the EXIMU program. Alarms and conclusions, predictions and brief health reports, are made suitably apparent.

IMU History Window

This is a window on the IMU History database, used for "what-if" analyses by the user, not covered automatically by rules in the IMU Analyst. The rules themselves can be viewed by windows on the Measurement Dictionary database.

Agent Windows

This is a window on the inferencing mechanisms, invoked in the case of diagnosing (if a fault is detected) or explanation (if status is concluded to be healthy). The user may manually override some of the actions (for example, explicitly requesting certain measurements by re-setting the state of the scheduler) as a safety precaution in case the inferences are faulty.

IMU Model Window

If model-based reasoning is employed, then this window would display the (hierarchical) schematics of the IMU cluster, datacomm system, etc.

4.5.1.5 K-Bus Design For Problem

Most of the objects identified above have an obvious K-Bus analogue. The EXIMU system as whole is an **application** consisting of three **agents** and other objects. Nothing more need be said about the user interface or distributed database management system. As mentioned above, the measurement source exists only for simulation purposes: its interface (initialize, start and stop) specifies the commands it responds to but its internals are of no interest here. Although the measurement stream could be viewed as an object, or stream of "message" objects, that is unnecessary for reasons of efficiency and because its only interface is with the device driver. Other objects are realized as follows:

Measurement Driver

This is a **driver** object, which read signals from the serial port, writes characters into its (private) buffer, and presents to other objects a sequential file interface. It is:

opened in **character mode** at the start of the program

read from

closed at the end of the program run

by the measurement preprocessor. For control of the measurements sent, it will:

write

to the serial port on commands from the request postprocessor.

Measurement Preprocessor

This is an **abstract sensor** object which reads from the measurement driver and writes records into its (private) buffer. It is:

reset and setup with parameters specifying the measurement driver, imu operational mode, and data reduction algorithm

started so as to be in continuous mode

read from

halted at the end of the program run

Several sensor objects may be needed to avoid loss of information in changing context when the operational mode changes.

On initialization it knows how to set up its internal tables etc. to translate the measurement input characters to records:

(<meas id> *start* <text string> *stop*) --> (<meas id> <time-tag> *pass/fail* <value> <imu> <subsystem>)

These records are written to the real-time measurement **DBMS**. The translation involves tokenizing the input characters and parsing using a lookup table constructed on initialization by subsetting the measurement dictionary **DBMS** according to operational mode. The time-tag is simply another measurement, sent once a second, so it is cached and copied - there is no need for an internal clock here. The filtering, scaling etc. is done by the specified data reduction algorithms as another pass in the record construction.

It may be desirable to initialize the sensor with a set of alarm criteria, of the form (<meas id> <conditions> <alarm id> <actions>), so that if the specified measurement satisfied the alarm conditions, then the specified alarm will be constructed (just like another record, with time-tag and any other relevant information). It could then be detected by an attached **probe** which would take the actions specified in the alarm , thereby short-circuiting any higher level processing.

Request Postprocessor

This is an **abstract effector** object which sends measurement request commands to the measurement driver. It is:

reset and setup with parameters specifying the measurement driver and command syntax

commanded

started to send the command(s)

halted at the end of the program run

On initialization it knows how to set up its internal tables etc. to transform the control commands to streams of appropriate characters. This is done as described above for the measurement sensor for mode changes (using the measurement dictionary **DBMS**); in addition there are commands to start and stop sending measurement which return error codes to be checked by an embedded sensor and acknowledgement policies.

Notice Boards

Each may be implemented as a simple **blackboard** with associated **message managers** and **probes** installed by the interested agents. Each has information

added and **deleted** by only one agent

so they will not need a transactional manager to resolve concurrent write conflicts. The structure of each is established at construction time and not expected to change dynamically so there will be no calls of the **restructure** method.

Intelligent Agents

Each **agent** is:

installed at the program start

removed at the end of the program run

by the **application** object.

They communicate with each other through the notice boards by attaching probes which look for events of interest. The events are used to focus attention within the agent to a subset of its capabilities. The design presented here is a loosely-coupled distributed AI system where there is very little direct communication between the agents. The monitor and scheduler continue their operation even though the analyst is diagnosing a possible anomaly, so that if it turns out that it was a false alarm, no data or events are lost.

Each agent may be hierarchically subdivided into specialists, for example the analyst may consist of:

- Three identical modules, one per IMU
- A module per IMU subsystem
- A module for the datacomm subsystem
- A module per operational mode

These specialists would each be a **knowledge source** with its own knowledge base and inference engine, and would communicate with each other within the agent through a blackboard or some other organizational paradigm.

4.5.2 Assessment Conclusions

The previous example of K-Bus use illustrates several advantages provided by the K-Bus concept.

This is an evolutionary approach - new technology can be accommodated by the open architecture, but is not a requirement for building an application. In contrast, many proposed architectures depend on revolutions in physical hardware such as massive parallelism, wide-band & high-speed communication, etc. As the EXIMU example illustrates, the layered architecture has a clear separation between considerations of physical distribution and logical (or problem-solving task) distribution: this is an example of hardware portability. There are a few coarse-grained high-level tasks in EXIMU, but it is easy to imagine a massively parallel implementation with separate processors for drivers, sensors, effectors, knowledge sources (and even rules).

Each object comes from a library of reusable software classes and has a standard interface defined by the K-Bus. Thus much of the programming is at the high-level of plugging in, and interchanging, K-Bus components, in contrast to building each application from scratch. This not only encourages cost savings in development and maintainability, but facilitates easy prototyping and testing. A conservative estimate for EXIMU is that 60% of the development effort would have been saved if the K-Bus tools identified above had been available, and the subsequent benefits in accommodating new requirements during maintenance would be similar.

The concept of probes and blackboards provides a release from the traditional operating system's weak support for communication and cooperation between modules. For example, in Unix, pipes provide unidirectional, byte-width, temporary channels for point-to-point communication between processes. In contrast, blackboards provide broadcast, high-bandwidth, semi-persistent communication channels between objects; the channels can be queried (for example, by probes) and they preserve the structure of messages for pattern-directed invocation. An agent can install a probe on a remote object, unknown to that object, and monitor its behavior in a non-intrusive manner; a special case of this is when an agent uses a probe to monitor changes on a blackboard. The semi-persistence of the

communication channels means that if a problem occurred, it would be easy to roll back the control flow: thus fault tolerance is encouraged. The same capabilities also enable dynamic validation, with probes monitoring the transactions between objects in a testbed environment.

The EXIMU example illustrates how event-driven problems are naturally mapped onto K-Bus objects. The design for such problems follows a dataflow analysis, where the primitive signals are gradually transformed to symbolic conclusions as they pass through the communication objects in the data flow diagram. The control is distributed and loosely coupled, which also encourages fault tolerance - if some processor or network component crashed, the whole system would not cease operation.

Finally, it should be noted that the architecture provides these same benefits for purely conventional programs as it does for knowledge-based systems.

4.5.3 Recommended Priorities for Implementation

An analysis of the current state of the art in software suggests that development of the K-Bus concept is most promising for the objects in layers 3 and 4. Below layer 3 the services are provided by off the shelf products, which only need object-oriented glue (i.e., an interface class definition) to be integrated into the overall architecture. While research into newer and better operating systems continues, it is expected that many interfaces can remain the same and the technology will be seamlessly incorporated as it becomes available. On the other hand, the research into languages to describe generic tasks, domain objects and generic applications is still emerging and may be too high a risk for an initial development effort. As a result, the generic objects in layers 5 and 6 can be viewed as placeholders which may not be implemented until prototypes of the lower layers are complete, and the technology behind these more abstract layers is better developed and understood.

In layers 3 and 4 there are a number of commercial products, most notably database managers, user interface managers, knowledge representation schemes and inference engines. Again, all that needs to be done with these is to provide an interface wrapper. Considerations of verification and validation require static verification tools and the construction of a testbed for dynamic evaluation using probes. Thus, the objects which are of highest priority in the development of the K-Bus are as follows:

- Object class definition wrappers for off-the-shelf products
- Abstract Sensors & Effectors
- Distributed Communication Model Objects: Cloner, PostOffice & Finder
- Probes
- Blackboards
- Testbed

These objects are prominent in the usage scenario above, which is a good example of application requirements for ALS. It is therefore suggested that these categories are the most important areas of the K-Bus to implement, because of both their eventual utility and as a proof of concept.

To support the proof of concept goal, it is suggested that a selection of objects from each category is chosen for implementation, sufficient to support the Phase II demonstrations, rather than to attempt complete coverage in one particular area. Several subclasses of the ones specified in Section 4.3 will eventually be identified from prototyping experience, but a few particular instances will be sufficient for the demonstrations.

5. VERIFICATION & VALIDATION OF KBS IN ALS

Any developer of a computer program has the objective of producing a system that can be *validated*; that is, a system that correctly implements useful requirements. The accepted means to this end involves (a) *developing high-quality requirements* through analysis and/or prototyping and (b) *following a development methodology* that maximizes the likelihood of a valid system. *Verification* that the methodology has been adhered to correctly gives a degree of confidence in the overall structure and internal details of a system. Although "verification" and "validation" have separate definitions, the maximum benefit is obtained by using them together and treating "V&V" as an integrated concept (Wallace and Fujii, 1989). Thus, V&V is a single discipline for ensuring that the phases of development from determining requirements through final deployment and maintenance mesh together and result in a useful system.

This section gives requirements and guidelines for an ALS V&V methodology for knowledge-based systems (KBSs). A detailed methodology meeting the guidelines and targeted specifically at systems running on the K-bus architecture (see Section 4) will be developed in Phase 2 of this project. This section is not intended to be an overall review of V&V problems and approaches for KBSs; see for example Geissman and Schultz, 1988, Johnson, 1988, Rushby, 1988, and Stachowitz, 1989. Rather, this section focuses on V&V in terms of anticipated ALS software architectures, applications and autonomous operations.

This section is divided into the following subsections. Section 5.1 is a summary. Section 5.2 is a brief overview of V&V for KBS, presenting a digest of existing literature and some summary assertions. Section 5.3 frames the subject of V&V for ALS KBSs by considering the implications of autonomous systems, a critical class of applications where KBSs are likely to be used. From this, Section 5.4 follows with a description of some specific risks that need to be considered. Section 5.5 derives the requirements for a V&V methodology and suggests development methodology guidelines.

5.1 SUMMARY

Maximize validity and verification through:

- **Tailoring requirements**
- **Raising level of implementation**
- **Using prototypes in requirements and design**
- **Assembling systems from high-level components**
- **Automating verification**

A valid system depends generally on having useful requirements and then correctly executing each of the transformations that lead from the requirements (problem statement) through the design to the code (solution statement). The likelihood of doing each of the transformations correctly is enhanced if there is not a great difference in the level of abstraction of the requirements and the code. Code itself (i.e., a knowledge base in the case of KBSs) can be checked for a number of syntactic, logical and even semantic errors by automated tools. Using these ideas as the base, we propose that ALS knowledge-base system development should incorporate the following:

- *Maximizing validity* of systems by minimizing the gap between requirements and implementation, resulting in better (i.e., more likely to be satisfied and more likely to be useful) requirements and implementations. Not only does this enhance validity, it simplifies validation.
- *Tailoring the level of requirements* through
  a) top-down end-to-end analysis of ALS,
  b) using generic tasks and applications as model requirements, and/or
  c) using prototypes to generate requirements where the problem is not well-defined or is one of replacing a human expert
- *Raising the level of implementation* through the development of a library of generic applications, based on generic tasks including knowledge processing heuristics, basic knowledge and commonly-used modules such as standardized user interface components
- *Using prototypes to explore design spaces* where an implementation cannot be specified as a combination of library components or other known techniques such as mathematical models or procedural programming structures
- *Simplifying and improving verification* through automated tools built specifically to interface with the libraries of generic components and their host system (e.g., Unis, K-bus)

The above is a summary of the recommended development/V&V methodology for KBSs in ALS. The ALS V&V methodology (to be developed in Phase 2) will describe the details of how one ensures that the development methodology is correctly followed.

5.2 OVERVIEW OF KBS V&V

Most major software projects follow a general software development standard or life-cycle model such as the DoD-STD-2167A, or the Spiral (iterative and/or prototype-based) development model associated with Boehm (1988). Although there is controversy about the effectiveness of the linear software process model implied by 2167A (e.g., Gruman, 1989), and the model's focus is procedural programs not KBSs, 2167A is a useful way of characterizing the stages and documentation products of a software project. The set of decisions that must be made and documented is relatively consistent, whatever kind of life-cycle is involved. However, the *sequence* of activities in KBS development can differ greatly from the waterfall model.

Instead of top-down progressive elaboration from definite specifications as in the waterfall model, KBSs are often characterized by iterative refinement of requirements and designs, evaluation of prototypes, and the deferring of decisions; some how-to-solve-it decisions may be deferred even until run-time(!) in goal-seeking programs. These distinctions reflect partly essential differences between procedural and KBSs, and partly the fact that KBSs are frequently called upon to solve hard, poorly-understood problems (Linden, 1988, 1989).

Conventional software life-cycle models like 2167A are only partially useful for knowledge-based systems.

The processes and documents in conventional models are generally appropriate to KBSs, but the implied sequence is likely to ignore iteration, prototyping and goal-seeking programming.

Development standards cover topics such as how to identify systems, what are the stages of development, and the forms of documentation that should be produced after each stage, including requirements, high-level

and detailed design, code, test plans and test reports. Commonly there is a stress on tracing from each level to the next lower one and inhibiting changes to requirements and basic design concepts once established. These models are generally applicable whether for procedural or KBSs, and whether or not prototyping is used, except that with prototyping the notion of change affecting requirements and basic design concepts is inescapable. How-to directions, where they exist, are generally implicit in the documentation standards.

However, because the guidelines have such broad application they cannot give much specific guidance for how to perform and verify the transformations that development consists of, other than to insist that this be done as completely as the documentation permits. A literature has developed addressing the specific needs of KBSs, stating needs or giving suggestions and guidelines for design, verification and validation.

The current literature on V&V for KBSs consists of papers that follow one of the following major paradigms:

- requirements or overviews of proposed development methodologies for KBSs. Johnson, 1988, is a good example of requirements for a methodology.
- taxonomies of the kinds of knowledge in KBSs, kinds of error that can creep in, and some ways to check for them, implying V&V guidelines. These papers can be interesting and frequently propagate valuable experience, but do not constitute a methodology.
- extensions of the previous category to include metaknowledge, or generic knowledge-processing task knowledge. Some papers by Stachowitz, et. al. extend this kind of discussion to a tool (EVA) that automatically checks knowledge base structures both for internal consistency and against metaknowledge describing the KB's putative tasks.
- papers that summarize V&V techniques for procedural systems and say that knowledge-based programs should, in principle, be treated the same way: get good requirements, have a development methodology, and follow the methodology.

One characteristic of this literature is that it commonly addresses the traditional concept of *expert systems*, which are systems constructed to assist or replicate a human expert in the performance of some task, frequently a pre-existing non-automated task. The theory of artificial intelligence implicit in expert systems as they are known today is rule-based inferencing using knowledge representations such as production rules, frames or semantic networks; connectionist approaches such as neural networks are still too close to the frontier and have not penetrated the V&V literature.

The applications for which expert systems are used frequently suffer from vague specifications and weak design methodologies, which can be serious obstacles to V&V. On the requirements side, precisely what human "experts" actually *do* can be hard to pin down. On the design side, the important subjective aspects of human experts' organizational and social interfaces are frequently overlooked in normal requirements gathering, and only tangible aspects like approval sequences are included; with no requirements it is hard to design these interfaces into systems. This helps to explain the emphasis on prototyping to refine fuzzy requirements and after-the-fact knowledge base checking tools and testing strategies, as opposed to the up-front use of specification and design methodologies/tools and problem-solving architectures.

When a system is conceived as an "expert system," specification and subsequent validation focus on the expert's knowledge. For example, consider the view of KBS validation that holds that it "must pragmatically be limited to insuring that executions of the implemented code reflect the knowledge, and nothing but the knowledge, intended by the person(s) from whom the knowledge is obtained. This knowledge must be captured during knowledge acquisition." (Goodwin and Robertson, 1988) This implies a concept of KBSs as knowledge, not as functions, and overlooks the notion that a KBS can be valid by virtue of adequately carrying out an assigned role in some larger system or organization. We will argue that the second view of validity—a system is valid if it carries out a role derived from an end-to-end analysis and

design—is more appropriate for ALS systems, and that KBS development and V&V methodologies used in ALS should reflect this.

In summary, the KBS literature contains descriptions of tools, individual techniques, and guidelines for the development of methodologies, but not methodologies *per se*. However, in a slightly different vein, Linden, 1988, argues that there are some inherent features of KBSs that V&V should be able to use to advantage, including the following:

- KBSs involve declarative knowledge representation and techniques for reasoning about it, thus enabling automated reasoning processes designed to verify and validate the knowledge (see the discussion of EVA, below)
- KBSs can use techniques for reasoning about uncertainty which can make software robust in the face of errors
- simulation techniques allow system validation on a wide variety of cases
- automatic program synthesis techniques, also known as inductive knowledge acquisition, can bypass some steps of knowledge base development or programming in a consistent and relatively reliable way—see 5.5.3.6, below

There are features unique to KBSs that V&V can exploit.

Existing KBS development methodologies are not necessarily efficient or integrated. By efficient we mean a methodology that is based on large-scale standard building blocks rather than individual rules or statements. An efficient methodology should permit one to build and V&V a system through the assembly of a small number of high-level components and a minimum amount of domain-specific knowledge. This parallels the way in which higher-level languages require fewer statements and the statements relate more closely to the problem being solved (e.g., FORTRAN is a natural and efficient language for describing mathematical operations, compared to assembly language; a subroutine library is more efficient; and a statistics package may be even better). Building KBSs from scratch every time is the equivalent of assembly language programming: it is possible and it can be verified, but it is inefficient and expensive and should only be called for in special cases.

An integrated methodology is one appropriate equally to procedural programs and KBSs. There are plenty of effective tools and guidelines for checking the consistency and "correctness" of a set of rules that operate near the level of a compiler's syntax checker, and these tools should be used. What are missing are tools and guidelines that work at the higher semantic level to assure that the syntactically correct and non-conflicting rulebase will in fact solve the problem it is intended to solve. Our recommended approach to this problem is a layered architecture of generic modules such as the K-bus (see Section 4).

V&V is based on development methodology.

V&V cannot make a product any better than its development methodology.

Verification confirms that the product was "correctly" built.

Verification is more difficult when there is a large gap in abstraction between requirements and implementation.

Verification is more difficult when there are many levels of abstraction between requirements and implementation.

Validation confirms that the product is "useful".

The relation of V&V to software development can be briefly summarized in the following assertions:

- a V&V methodology is built around a development methodology
- validation cannot make a product any better and cannot be more definite than the product's requirements, and validation failures are the result of poor requirements as often (if not more often) as poor implementations
- a development methodology divides the conversion of a high-level, abstract (problem domain) requirements statement to low-level concrete (computer domain) code into a number of stages with specific inputs and outputs; each stage is a transformation that makes the solution more concrete
- verification (by definition) confirms that the transformations have been carried out correctly according to the methodology
- there is a verification-development tradeoff between few, large transformations and many, small ones:
  - verification is more difficult when a transformation bridges a wide gap in levels of abstraction
  - verification is more difficult when there are more level transformations
- verification cannot make a product any better than the development methodology that went into it
- a verification methodology describes specific artifacts and techniques for tracking the development methodology
- advances in the verification process are largely improvements in artifact-creation during the development process
- reliable automation of stage-to-stage transformations in the development process (e.g., converting a specification into a design) adds repeatability and consistency to those transformations and can greatly simplify their verification

From these assertions we conclude the following objectives for a development methodology and corresponding V&V methodology:

Lower the level of requirements through prototypes, generic application models and end-to-end analysis.

Raise the level of implementation through libraries of generic tasks and applications.

Generate required documentation automatically.

- requirements for individual subsystems should result from end-to-end analysis of the overall system, system design and subsystem allocation, rather than case-by-case analysis (i.e., large systems should be developed top-down rather than through opportunistic bottom-up integration)
- there should be a minimum gap in level of abstraction between requirements and implementation: the level of requirements should be lowered, and the level of code should be raised
  - the level of requirements is lowered where a system is defined as fulfilling a subsystem role within larger systems that are designed from the top down. In this case, the system has to emulate a paradigmatic role in the abstract (or implemented) higher-level system. (This adds a layer between the top-level "ideal" requirements and the requirements that developers are immediately aiming at.)
  - alternatively, the level of requirements is also lowered where a system is defined in terms of a prototype. In this case, the system has to emulate the prototype.
  - alternatively, the level of requirements is also lowered where a system is defined in terms of an abstract system or task that is is an instance of.
  - the level of implementation is raised where pre-tested modules are provided that represent generic applications and generic tasks; this also has the effect of reducing the number of interfaces (fewer modules than lines of code), which simplifies testing.
- there should be a minimum number of transformations required to bridge this gap
- the process of bridging the gap should automatically generate formal documentation or other artifacts suitable for verification

The objective, then, is not to propose incremental improvement to current development and V&V methodologies to fill in this or that perceived gap. If a system the size of ALS is fully automated (i.e., is intended to be fully autonomous), there will be too many interfaces, too many lines of code for any piecemeal approach to make sufficient improvements. In any case, existing *techniques* are not the problem: solutions exist to the technical problems of V&V; the problem is that software requirements analysis, development and V&V are too often applied after all the other constraints have been determined rather than having a fundamental role in overall system design.

An approach is proposed for KBS development—and the corresponding V&V—based on libraries of standard modules rather than individual lines of code. This approach is common in procedural programming, where function libraries are employed for standard tasks like mathematical analysis, database access and menu management. It is not, however, common practice in KBSs or in mixed knowledge-based and procedural systems.

5.3. V&V ISSUES FOR AUTONOMOUS KBSs

Software V&V is an especially critical issue in ALS because of the high degree of autonomy planned. The ALS Technical Reference Document states that "[a]ll vehicle applications software will function autonomously with the exception of range safety." Programs operating autonomously will have to meet special safety and reliability requirements and their V&V program has to ensure that these requirements are met. (Section 3 of this report is a detailed examination of implementing autonomy in ALS.)

An implication of this policy is that all activities may be computerized, and "It's too hard, let the human do it" may no longer be an option. (Systems engineering trade-off analyses will have to determine exactly what is automated initially and the rate at which the human-operated residual is replaced. Obviously, with today's technology most activities will continue to have a man in the loop, but where humans persist, they should view themselves as requirements-gathering prototypes for automated systems and organize and document their activities accordingly.) Thus, the software development methodology will have to be extended beyond what can be achieved reliably with the standard procedural paradigm.

One consequence of autonomy is that programs will probably be required to do things beyond the normal state-of-the-art in KBS development. Four kinds of programs, at least, are relevant from the point of view of levying autonomy-related V&V requirements; any one ALS sub-system be in more than one category:

- Programs that must absorb errors such as hardware malfunctions and continue normal operation without impacting other components they are connected to. Examples include on-board GN&C (post-launch) and ground-based test equipment (pre-launch). Keywords are reliability and fault-tolerance.
- Programs that control mission- and safety-critical operations and consequently make mission- and safety-critical decisions. Keywords are safety and reliability.
- Data-driven or goal-driven programs with large potential problem spaces where exhaustive testing may not be feasible. Keywords are deadlines, real-time and self-analysis.
- Distributed systems, where each component can operate only with the cooperation of a number of other components, and where the underlying operating framework (e.g., network or message-passing functions, distributed DBMS, distributed blackboards) is a significant yet possibly invisible element. Keywords are critical dependencies and reconfigurability.

V&V intersects autonomy through the specification of a design methodology that anticipates the particular needs of autonomous components and verifying that the design methodology was followed. Generally, the design implications of autonomy are that faults and failures must be contained locally and not propagate through the system. This is not to insist that all failures must be corrected locally; in some cases a controlled shut-down is acceptable ("fail-safe"), where the system abandons autonomy to alert its operators and the systems it communicates with before stopping. (This implies a definite upper limit or envelope around autonomy in risky situations.) Therefore design must include postulating failures, and testing must include causing failures and ensuring that there are no detrimental external effects. The trade-off between building a fail-safe system (that recognizes errors and stops) and a fault-tolerant failure-correcting system (that absorbs errors and continues) must include the difficulty of verifying that all potential failures are, indeed, correctly dealt with. For certain mission phases (e.g., 15 seconds after ignition) there may be no alternatives to fault-tolerance.

To permit autonomy, V&V must ensure that safety objectives are met.

The general nature of risks from computer systems is well known. These can be serious in individual systems, and are very sensitive to multiplying where several components are involved, as in the anticipated ALS autonomous architecture case.

As well as simple mistakes by developers and all the kinds of errors common in procedural systems, KBSs have special features of their own. Errors often stem from KBSs' tendency to exhibit shallow recipe knowledge, which results in not recognizing errors and stopping before things get worse, and not knowing when the realm of competency has been left behind. If KBSs are based on copying human procedures, they may also introduce the kinds of errors that humans are known for: they are inconsistent, not exactly reproducible, capable of being overwhelmed by data, often ignorant of relevant data, slow, expensive to create and maintain, and subject to extensive down time. Fortunately, however, many of the humans' shortcomings are exactly in the areas where computers are strongest. On the other hand, humans have common sense, which helps guard against exactly the kinds of errors attributed to KBSs and other computer systems.

Divide tasks appropriately among humans, conventional systems and knowledge-based systems.

As the first step in scoping the use of knowledge-based software in ALS, consider the following general allocation of tasks to humans, procedural programs and knowledge-based programs. (See the Maximum Autonomous Architecture report, Section 3, for a more complete treatment and evaluation against ALS system requirements.)

- *humans*: take responsibility and sign-off major decisions; monitor and back-up automated systems; deal with unpredicted circumstances; tasks that do not otherwise fall into one of the following two categories.
- *conventional programs*: deal with real-time requirements such as measurement, data storage/retrieval; handle problems for which programs are currently employed, and other well-formed problems (that might also be suitable for knowledge-based programs) where requirements are unlikely to change; provide prototypes for requirements analysis.
- *knowledge-based or inference-based programs*: deal with tasks that are instances of well-understood reasoning skills such as (after Chandrasekaran, 1986) hierarchical classification, hypothesis matching, information passing, plan selection and refinement for configuration synthesis, state abstraction and abductive assembly of hypotheses. Examples of these are major elements of the ALSYM system model; mission planning; fault diagnosis; classification; configuration layout; and monitoring performance and replanning as needed.

Note that a general category of "expert systems" to carry out tasks migrated from humans is *not* included. Specific functions are assigned to humans and others to programs of various types (although initially humans will probably stand in for some of the computer systems). In time, the boundary may move to enlarge the scope for automation by replacing humans and KBS technology can be used for decision-support for the humans, which would lead to opportunities for expert systems. The reason that "expert systems" *per se* are missing is that the automation of ALS can be planned in advance, rather than growing up around a manual technology.

From a technical feasibility perspective, conventional or knowledge-based systems can be virtually *anywhere* in the overall system. The constraints are issues of availability of inputs and paths for outputs ("hooks" need to be built into ALS from the beginning), and the decision between a KBS or a procedural logic box is purely one of appropriateness for the particular application (and procedural components can later be removed and replaced by KBSs or *vice versa*).

5.3.1 Current Implementation Options

Today, a significant choice in software implementation involves deciding between a procedural or knowledge-based implementation. (See Section 2 for a methodology to assist in this decision.) The factors in the trade-off include how well-formed the problem is, the nature of the solution techniques required, and the anticipated frequency of maintenance and technology upgrades.

Poorly-formed problems may require a prototyping solu–tion, which could be either procedural or KBS. From the prototype it can be determined which way to go for a delivery system.

Heuristic, data-driven or search-based solution methods are sometimes easily implemented using KBS tools, yet too complex for procedural solutions.

An architecture like that in KBSs that separates explicit control knowledge (the knowledge base) from the pro–cessor (the inference engine) can be easier to maintain than many procedural design strategies where control and processing are implicit and intertwined. This can be significant in long-lived systems like ALS.

Detailed design guidelines follow from matching the above generalized three-way human-procedural-KB breakdown to the maximum autonomous architecture (see Section 3) to give a set of tasks for which KBSs are development alternatives that should be considered in development trade-offs (see also ALS Technical Reference Document, Section 4.4.3):

- fleet and multiple mission planning
- single mission planning and high-level status monitoring,
- vehicle configuration,
- manufacturing planning,
- inventory control,
- manufacturing control,
- manufacturing test and diagnosis,
- vehicle assembly and checkout planning,
- vehicle checkout,
- fault diagnosis and fault recovery planning,
- launch sequence planning,
- launch sequence control,
- mission status monitoring (situation assessment),
- adaptive on-board avionics,
- other on-board control systems,
- reporting on all of the above,

- what-if hypothetical analyses based on all of the above,
- displaying current status of all of the above,
- (everything except range safety).

Developing the above knowledge-based applications implies a widely distributed system, as these applications necessarily operate at different times, on remote hardware, have frequently unrelated knowledge bases, and would overwhelm a single host. A distributed system implies increased V&V sensitivities as the number of individual components and the number of interfaces increase, and the distributed host comes into play. The major issue from the sensitivities perspective is the number of interfaces, because run-time surprises are more likely as interfaces multiply. Each component should have few interfaces, and where possible the interfaces should all be to a single (distributed) host or platform rather than a number of individual interconnected hosts. In this way functions to protect modules from bad data and prevent errors from propagating can be centralized in one subsystem, enhancing the development and verification of these functions.

5.3.2 Potential Connectionist Implementation Options

In the future, connectionist AI may contribute to the solution to the fault-tolerance problem, as well as enabling solution of new problems of intelligence. McCullough and Pitts (1943) proved that networks of formal neurons are computationally as powerful as Turing machines, and von Neumann (1956) and Winograd and Cowan (1963) studied the role of redundancy and the distributed representation of information in the problem of constructing reliable nets from unreliable neurons. At this point in history, however, expert systems and the associated V&V practices focus on rule-based symbol processing, and that is what is addressed in this report.

5.4 RISKS AND RISK MITIGATION FOR KBSS

A special class of reliability is the avoidance of risks. The requirements for procedural and KBSs both will be severe in this area. Current practice with procedural software has led to the following observation:

> "...current software reliability figures are, at least, orders of magnitude less than required.... One option is not to build these systems or not to use computers to control them. The current rush to use computers to control nearly every type of device may involve a seriously unrealistic discounting of the potential risk.... There are, however, systems where a realistic risk-benefit tradeoff might conclude that computer control is justified. In these cases, a non-absolute approach to reliability may be possible.... It seems reasonable to devise techniques that focus on those failures with the most serious consequences....
>
> "Even if all failures cannot be prevented, it may be possible to ensure that the failures that do occur are of minor consequence or that even if a potentially serious failure does occur, the system will 'fail safe'...." (Cha, Leveson & Shimeall, 1988)

Risks are considered in terms of safety, reliability and security. (Software development risk—the risk that a project will not be successful—is a separate topic; for a discussion of a risk-avoidance strategy based on the spiral model life-cycle, see Boehm, 1988.) Safety is the avoidance of hazards, events which can endanger a mission, people or material (Leveson, 1986). A hazard is not a class of software error, rather it is a possible outcome of a software (or hardware) error. Note that risk potential is not principally a function of how a system is implemented, except insofar as a particular implementation is known to be less than perfectly reliable. Rather, risk potential results from what a system does: a system that sorts mailing labels by ZIP code is almost by definition not a potential risk no matter how it is implemented. Effecting risk mitigation, however, may be implementation-specific.

The magnitude of a risk is modeled as the combination of hazard seriousness and likelihood of occurrence (which is the inverse of system reliability). Hazard analysis can estimate the magnitudes and likelihood of

specific failures propagating to specific hazards, and reliability models can be applied to estimating the frequency of occurrence. Although reliability modeling is well-developed for hardware and conventional software, it is a relatively new issue with KBSs. Reliability modeling is tricky with KBSs because (a) there is uncertainty concerning what constitutes a failure, (b) there is a major profile shift between testing and operation, and (c) KBSs typically undergo continuous change or evolution (Rushby, 1988). However, Barrett, 1989, believes that despite these problems "software reliability models can be applied to expert systems," and proposes research efforts to develop and enhance appropriate models.

Reliability raises the issue of redundancy, which can be in design, in heuristics, in hardware, or in multiple independent systems, including human oversight. Additional complexity arises where redundant decision recommendations are made, for yet another decision must be made to select one recommendation to be acted upon through a mechanism like voting, so that additional computer resources are required.

There is little question of *technical risk* from the perspective of whether or not the KBS paradigm can meet functional requirements. Existing KBSs that perform mission monitoring and control (e.g., KATE), configuration (e.g., EXCABL, R1), fault analysis and diagnosis (e.g., IN-ATE), and navigation (e.g., NAVEX) demonstrate that knowledge-based methods are generally feasible for a variety of applications.

5.4.1 KBS Risk Mitigation Requirements

In a KBS, dealing with risks can be a particular problem because of the priority accorded risk avoidance. Good design indicates that knowledge bases should be designed to deal with abstract tasks, rather than having to constantly worry about application-specific issues, yet there needs to be a way to locate knowledge of how to identify and avoid potential risks within the system at appropriate places. An analogy is the set of human involuntary reaction mechanisms that respond to bright light (pupils), stress (adrenalin), and so forth with appropriate responses without requiring conscious thought. For some subsystems real-time performance requirements will be an additional complicating factor here because risk avoidance has not only a high logical priority, it also may have an absolutely high temporal priority: if a potential serious hazard occurs, the ALS system must be prepared to "drop everything" in some way and deal with it. Leveson, 1989, observes that "safety is enhanced if the risk of accidents is reduced, even though the process of risk reduction may require the (perhaps temporary) non-satisfaction of some or all of the functional or mission requirements of the software or of the encompassing system."

A reasonable approach would seem to use a layered architecture, with low-level components like sensors and effectors containing some of the risk detection and avoidance implementation, including the real-time critical elements. For example, consider an effector to perform a potentially hazardous task such as igniting a rocket or opening a valve to start a toxic material flow. The effector can be blocked against performing this operation until an all-clear state is declared by a KBS that is planning and controlling the entire complex operation; if the KBS is likely to take some time to determine whether the all-clear state has been reached, then the overall sequence may have to be planned to allow for small delays between steps. Similarly, the effector can be linked directly to a nearby sensor and programmed to close the valve in case of a leak without waiting for instructions from the KBS (the sensor and effector both also report to the KBS, which will plan what to do next).

5.4.2 Procedural vs. KBSs

A treatment of risk mitigation for KBSs should focus on what is different about KBSs compared to procedural systems. Leveson, 1986, covers the field well and suggests that "[b]y examining why adding computers seems to...perhaps increase risk, it may be possible to determine how to change or augment the current techniques." Here we will consider what could happen to increase risks as KBS components are added to a system. If there are no distinctions between KBSs and procedural components, then existing methodologies probably apply; however, as is the case, a few additions will have to be made for KBSs.

5.4.2.1 Knowledge Based or Inference Based Systems

Knowledge-based systems' general characteristics result from the fact that processing control lies with an inference engine, and its strategy or paradigm is implicitly an important parameter in the system design.

However, the inference engine may be to some extent influenced or overriden where processing control appears explicitly in the knowledge base, making the knowledge base a mixture of what, why and how.

A knowledge-based solution may seem relatively more procedural (paradigms such as forward chaining, scripts or frames with attached procedures) or relatively more declarative (paradigms such as semantic networks). A given knowledge base, however, may be indeterminate, such as IF-THEN rules, which can in principle be used equally well by forward- or backward-chaining inference engines (and the NEXPERT inference engine does just that). In any event, the flow of control may not be visible, and the strategy may change from one paradigm to another without explicit programmatic direction, depending on the inference engine. The underlying inference engine is typically a procedural program (e.g., CLIPS), a symbol processor (e.g., implemented in LISP), or PROLOG.

Common problems with knowledge-based systems include fragility, unpredictability, brittleness and discontinuity.

There are several kinds of errors to which KBSs are prone. The following are some general categories (from Bundy, 1988):

- Fragility (non-robustness): the system may fail in unexpected ways
- Unpredictability: The user cannot specify the circumstances under which the system will produce an answer or cannot specify the type of answer that will be produced
- Brittleness (non-flexibility): The system cannot deal with problems on which it has not been previously used
- Discontinuity: The system gives very different output in response to similar input

These kinds of errors contrast with the more traditional branches of engineering where even when mistakes are made, outright surprises are few. The broad categories are made up of specific types of errors including the following important ones:

- inference engine errors (procedural execution) or incorrect or incomplete paradigm or language specification (e.g., handling of null values; initialization of working memory elements; etc.)
- incorrect (or non-optimal) "facts"
- incorrect (or non-optimal) problem-solving strategies
- conflict between knowledge and strategies for using the knowledge
- logical errors, redundancies, loops, and so forth within the problem-solving strategy
- leaving the system's realm of competence (the range of input data where the knowledge base is meaningful)
- unexpected side-effects because of global data used to link related sub-problems
- no restrictions on leaving the realm of available data (e.g., creating unattainable goals, issuing unexecutable commands)
- explicit control conflicts with implicit paradigm, lowering accuracy or efficiency (e.g., writing a procedural program in PROLOG)
- uncontrolled exploding search space leads to failure to meet constraints like time
- thrashing as mounting sub-goals prevent main goals from being dealt with

- conventional errors in embedded or attached conventional components

Some of these errors are relatively straightforward, such as logical conflicts within a knowledge base, and can be found by automated tools in a static analysis. Others are more subtle, such as not constraining a system to the subset of a problem space where it is competent, and may not show up in thousands of dynamic test cases, although a competence boundary can sometimes be inferred where a KBS becomes very sensitive to small changes in inputs when it nears it. Some other inconsistency errors (e.g., data interpreted in different ways in different places) have much in common with errors in procedural programs. Distinctive to KBSs are certain kinds of discontinuity, the potential for an exploding search space and indeterminacy thrashing; procedural programs are more likely simply to run to the same conclusion, right or wrong, every time.

Where there are systematic reasons for problems with KBSs, they can be attacked by modifying the development methodology. Bundy (1988) has identified some systematic reasons:

- The mixing of control and factual information within the same rule
- The attachment of arbitrary procedures to rules
- The use of multiple knowledge representation formalisms without a clear understanding of their relationship to each other
- The use of "uncertainty factors" without a clear understanding of their meaning
- The incremental development of systems by patches and hacks, and without consideration of the whole system
- A lack of theoretical understanding of the techniques used in the system's development

The development methodology suggested here attacks these issues.

5.4.2.2 Conventional Components

The characteristics of conventional or procedural components are well known and will not be considered here in much detail. The principal contrast with KBSs is that processing control is explicit—the program is a statement of how to do a task. The designer's underlying knowledge of the task that led to the chosen algorithms, however, is hidden and merely implicit. If the underlying knowledge is itself relatively shallow and procedural (first do this, then this,...), there may be a 1:1 mapping between the knowledge and the program, although this is rare. Typically other kinds of knowledge and implicit tradeoffs underlie algorithm design and are not visible in the code nor documented.

The kinds of errors that can occur and the corresponding risks are many. For example, every issue of *Software Engineering Notes*, the publication of the Association for Computing Machinery's special interest group on software engineering, contains dozens of examples.

5.5 V&V REQUIREMENTS

This section presents the requirements for a detailed V&V methodology. Because the verification half of V&V involves comparing development procedures with a methodology, these requirements also imply the requirements for a development methodology.

Specific procedures defining a methodology that meets these requirements need to be tailored to individual development environments and applications. Following this project, Phase 2 includes the specification of knowledge acquisition and V&V methodologies for applications developed under the K-bus architecture and object library.

Because quality cannot be verified or tested into a product after it is built, there is a stress on three aspects:

- Requirements definition that results in applying KBSs in appropriate well-scoped applications,
- the development methodology phases, including recognition of the role of prototyping and iterative development
- hosting systems on a distributed platform that ensures interfaces and data quality, and implements methods for cooperation in distributed systems permitting autonomous operations throughout ALS

An underlying assumption is that all tools, such as compilers, inference engines and other development and run-time software, will be certified. This is normal practice with procedural system development where ANSI standards exist for languages and ANSI and DoD assist in the certification of compilers, but heretofore this has not been a standard for KBS tools.

5.5.1 Summary of Development Requirements

For well-formed problems, develop top-down;

For less well-formed problems, use prototypes;

In either case, build applications from a library of verified generic components.

5.5.1.1 Well-Formed Problems: Assembly from Generic Tasks

Ideally, an end-to-end systems analysis will result in a number of well-defined problems, where KBS is an implementation alternative to procedural code. By well-defined, we mean that requirements can be stated for the inputs and outputs, and the objective of the component. Requirements can be especially "firm" where there is little user interface other than to monitor operations and provide some parameters, as in most autonomous operations. (User interface-intensive applications, on the other hand, such as decision support systems, are almost by definition not in this category.)

For example, consider an assembly planning function. The requirements are relatively straightforward, and can be stated as inputs, outputs and processing functions. Inputs, from Unis, might be a projected payload mass and shroud requirements, an inventory of available launchers, and launch window and orbit requirements; the outputs, directed to Unis, might be a plan and schedule for assembling and testing the stack. Conventional requirements specification techniques are fine for the inputs and outputs, but are likely to fall down is in describing the processing functions of KBSs. The suggested improvement is to define a set of generic applications (e.g., planning) and knowledge processing tasks within those applications (e.g., configuring under constraints), and to define the requirements in terms of these.

To explain the notion of generic applications and tasks, consider the analogy of programming a multivariate statistical analysis application such as factor analysis. This application can be defined as a sequence of generic mathematical tasks, including matrix inversion, eigenvalue extraction and matrix multiplication. However, a programmer would never consider writing code to perform these tasks; instead, he would either use a subroutine library containing the generic tasks, or a statistics system containing a generic version of the entire application. To write code for the tasks simply wastes time and invites error: the code in the library or statistics system is already verified (one hopes), so verification demands confirming only the interfaces to it. (Validation will still need to be performed because the input data, for instance, might be wrong.)

In cases where KBS is the chosen implementation, each component should be built according to the following guidelines:

- Identify and specify the *generic applications* in the above list of applications. Include all global data, interactions between subcomponents, and valid data ranges. This is analogous to a spreadsheet template, which can perform a particular application (e.g., calculating and printing income tax returns) given only a few parameters (e.g., income, deductions) and a spreadsheet interpreter program (e.g., 1-2-3).

- Identify and specify the *generic tasks* in the set of generic applications (see Chandrasekaran, 1986, for a core set of six key tasks). Include all global data, interactions between subcomponents, and valid data ranges. This is analogous to a spreadsheet interpreter program like 1-2-3.

- Identify and specify the *specific heuristics and algorithms* used within the generic tasks. Include all global data, interactions between subcomponents, and valid data ranges. This is analogous to particular functions within a spreadsheet interpreter, such as calculating present value.

- Develop a library of the specific heuristics and algorithms; verify the construction and resulting ranges of data validity. Each module should have limited and well-specified interfaces

- Develop a library of generic tasks from the specific heuristics and algorithms with limited other code; verify the construction from these components and resulting ranges of data validity. Each module should have limited and well-specified interfaces.

- Develop a library of the other common modules required in prototypes and systems, such as user interface, DBMS and network access. These modules will permit consistent look-and-feel and avoid hand-tooling in these important areas.

- Develop a library of generic applications from the generic tasks with limited other code; verify the construction from these components and resulting ranges of data validity. Each module should have limited and well-specified interfaces

- Verify each library element against its specification

- Develop a distributed platform to manage coordination and communications among distributed system elements such as sensors, effectors, database servers, KBSs and procedural applications. The objective is to reduce the number of interfaces that have to be checked out from N-squared to N in an N-component distributed application.

- The specification of new applications then consists of specifying the generic functions involved, identifying a set of generic modules and the additional application- or domain-specific customization knowledge. Verification of applications then consists of ensuring that the correct generic modules were selected, the customization knowledge is correct, and the customization knowledge is entered correctly.

- Although based on a specify-then-build model, developing new applications should include prototyping and iterative refinement *at each stage*: requirements, system-level design and knowledge base-level design. It is anticipated that the use of the high-level generic application modules will permit this without causing undue duplication of effort.

- A corollary of combining (1) specify-then-build and (2) prototyping or iterative refinement is that specifications and other documentation should be brief and to the point, because multiple versions may be required. Perhaps all knowledge of KBSs

can be concentrated in a single CASE system that can maintain multiple views of a system (requirements, generic tasks, system-level design, etc.)

- Validation of applications consists of extensive testing and evaluation of appropriateness. There seems to be no way around this, although the above focus on generic tasks in both specifications and implementations should reduce development effort and cause more systems to be valid.

5.5.1.2 Less Well-Formed Problems: Iterative Prototypes

Sometimes the statement of accurate and reliable requirements cannot be completed until users gain some experience with proposed systems (Luqi, 1989). When automating existing manual procedures, knowledge engineers may encounter a project where the assignment is to develop a system to either assist or replicate one or more existing human experts in a task. Frequently these are relatively fuzzy projects, where it is not initially clear what is to be done in both requirements and design. Requirements tend to be fuzzy where it is not clear which of the possibly many tasks the expert performs should be automated, where inter-expert interfaces are informal, where it is not clear how much (if any) re-design and change of existing procedures is acceptable (are we building a clone of an expert or a decision-support system to be used by an expert?), and/or where the problems, knowledge and interfaces the expert deals with change over time.

Not only are requirements likely to be fuzzy when automating an expert's tasks, but the topics that appear in requirements statements may be different from what is normally encountered. On the one hand, the "harder" requirements that apply to embedded and batch systems, such as response time or device interfaces, are often not relevant when automating an expert task, so that the developers may select what is appropriate from their viewpoint. On the other hand, some new kinds of "soft" topics may be introduced into requirements, such as "ease of use," "competency" (see below) and "ease of maintaining knowledge base." The precision of V&V will certainly be influenced by the fuzziness of requirements, and users should bear this in mind when judging developers.

Factors that make a design fuzzy include the expert not knowing what he knows (he just knows it) or knowing how he solves a problem generally (only specifically), and having the solution techniques evolve over time as experience is gained. Prototyping and iterative enhancement have become the accepted solution to this fuzziness.

Prototypes are normally developed in high-level languages that enable them to be built quickly and changed easily with little code, which gives the user rapid turn-around to his feedback. The purpose of a prototype is efficient use of user and designer or knowledge engineer time, and rapid feedback, not high performance or robust operation. A prototype should be easy to modify, because its requirements will be changing, and its code should be easy to read and analyze because it represents the developer's and user's understanding of the system (Luqi and Berzins, 1988). Figure 5.1 is a view of the prototype development cycle.

As discussed earlier, a prototype is an excellent aid to stating requirements. A prototype can be used in the target environment, and iteratively modified until it demonstrably fills a useful need and satisfies the users. Thus, even if the users did not initially know what they wanted, they can point to a prototype as a concrete expression of many of their requirements (or some departures from requirements, in case they do not like certain features of the prototype). As an aside, it is certainly reasonable to consider a situation where prototype KBSs are used to determine requirements, while the final delivery system is implemented in some other manner.

A prototype is also a useful way to determine an optimal design where it is unclear how to solve a problem, assuming that high-level tools exist that can be manipulated relatively easily. Different solutions can be tried until correct results are achieved and confidence is established that the heuristics work in all or sufficiently many cases.

The development cycle described in Citrenbaum and Geissman, 1986, and Miller, 1989, also explicitly uses prototypes in multiple roles, and allows for an altogether separate phase for conversion to a delivery system. The cycle starts with prototype-based iteration for requirements specification, followed by possibly

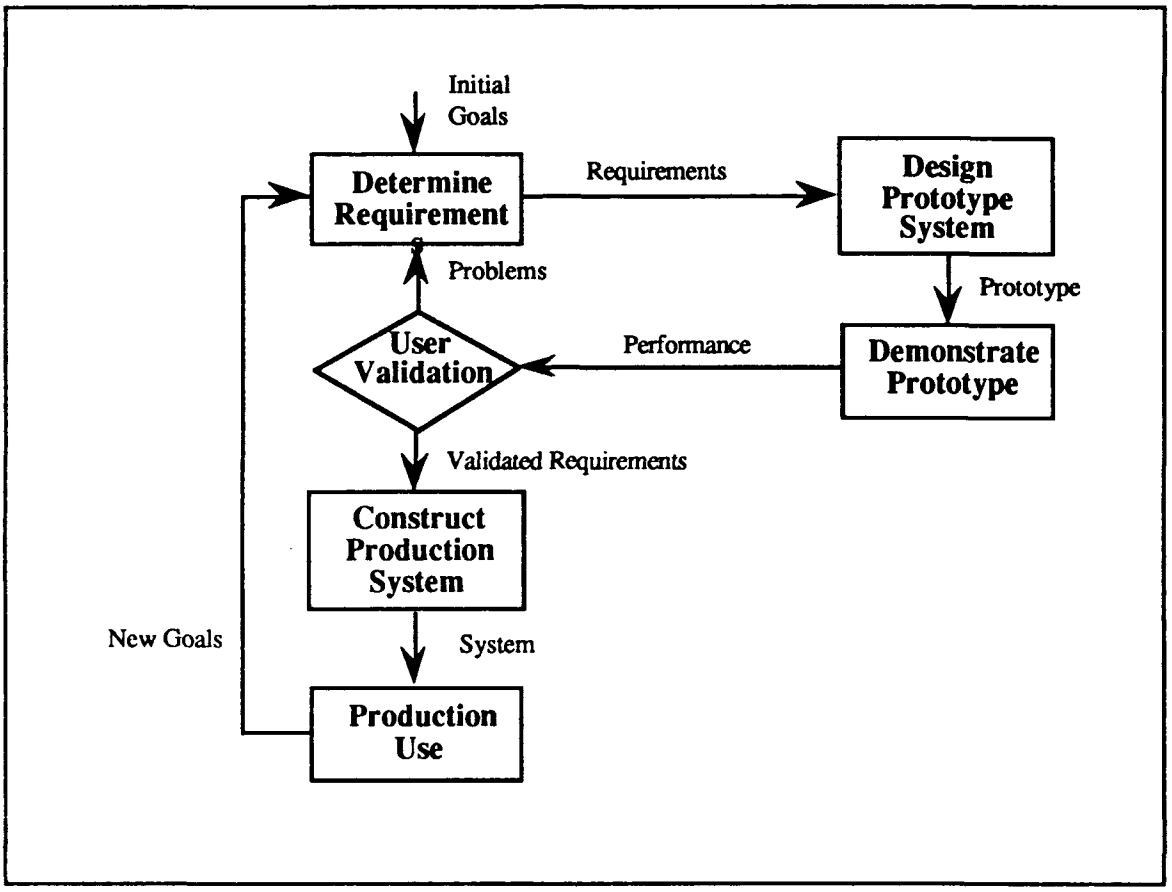


Figure 5.1—Prototyping and iterative enhancement have become the accepted solution to the fuzziness of software requirements. (after Luqi, 1989)

more iteration in the design or implementation stage, followed by possible conversion to a different implementation for the ultimate delivery system.

In the area of requirements, ALS will have scope for prototypes, but probably not as many opportunities as is currently the case when retrofitting automation into an existing system and organization, such as Space Shuttle. This is because there will not be initially a body of experts serving to solve routine problems and integrate the inputs and outputs of other automated systems: many of these roles will have been built in from the ground up through an end-to-end analysis. (The problems that experts deal with are initially thought to be highly specialized and not at all routine—which is one reason they are called experts—but the tasks frequently become routine in time.) Some experts will be in place, of course, and they may be gradually assisted or replaced by automation as time goes on.

In the area of design, however, there will be many opportunities to prototype ALS systems. Precisely because many KBS problems will be new and not already handled by human experts, there will be some uncertainty about the most effective means to automate. Rule-based? Frame-based? Numerical methods? Procedural? Prototyping, especially with generic applications and generic tasks, is an excellent way to explore the solution space of difficult problems.

In the inherently relatively fuzzy world of decision-support systems, some systems never evolve beyond the prototype stage and the prototype is delivered, because they need to be changed so often that it never becomes worthwhile to re-implement the prototype in another form.

When prototyping, the same modules mentioned above under "well-formed problems" should be used as building blocks for functions such as reasoning methods, user interface, DBMS and network interface, for

reasons of productivity and consistency. Where prototypes reveal new and general solutions to existing problems in these areas, the modules should be added to the relevant generic library. Figure 5.2 (from Luqi and Ketabchi, 1988) depicts prototyping using a library of modular, high-level components as building blocks.

Because prototyping involves iterative repetition of development, V&V procedures will also have to be repeated. To prevent workloads from becoming excessive, this implies that V&V should be automated wherever possible and that documentation requirements should be limited to the essentials.

5.5.2 Scoping KBS Applications

A development/V&V methodology cannot in itself ensure worthwhile products if systems are developed to unsound requirements or in inappropriate applications. The first step in ALS development is to define applications in an end-to-end analysis of ALS operations that goes from initial conception to routine operations and attempts to anticipate every interrelationship and interface. There will be little piecemeal after-the-fact development of systems that "do what Joe does," because there are no Joes there yet, hence ALS systems should have some of the best requirements yet devised. Instead, an integrated hierarchical structure can be devised initially with a granularity appropriate for development and minimal inter-system side-effects. (Because of technology evolution during the life of the system, the *implementation* should be one that lends itself to the replacement and/or integration of individual components or systems.)

5.5.2.1 Determining Application Boundaries

In ALS, KBSs will be there from the ground up (it's not a case of replacing and/or improving upon existing manual practices), so the standard problem of requirements indeterminacy should not apply. Expert systems are simply an implementation alternative for well-defined problem areas, and will have the same sort of requirements as procedural programs.

> **ALS will have few "expert systems" that automate the practices of an existing human expert. Instead, many KBSs will be in place from day one. Prototyping will be a tool to explore designs as much as requirements.**

Prototyping (such as under ADP projects) should be undertaken where it is not clear how to perform functions or allocate functions to subsystems. However, these prototypes should be used only to generate requirements and then thrown away. (Cf. iterative prototyping in requirements specification and development, above.)

5.5.2.2 Requirements Definition and Documentation

As suggested above, the basic statement of what a required system is should be based on either the generic application(s) and/or knowledge processing task(s) involved, or, where the problem cannot be stated in these terms, a prototype to be emulated. Generic applications and tasks represent a purposely restricted vocabulary specifically intended to clarify requirements and force the writer of the requirements to use these categories that are meaningful to an implementer. This is analogous to the situation with procedural requirements languages such as SREM (Alford, 1985) or SADT (Ross, 1985). Each of these languages is most appropriate for specific classes of applications. For example, within its "ideal" domain of time-constrained stimulus-response-type control systems, SREM's restricted vocabulary encourages the requirements writer to make specific statements meaningful to developers and testers, and impedes the development of fuzzy and ambiguous requirements. Generic knowledge processing applications and tasks (e.g., Chandrasekaran, 1986; see also the K-bus layered architecture in this report) have the same role for KBSs.

Both generic applications and prototypes are excellent languages for describing a system and guiding its implementation. However, it can be hard to derive tests from such requirements, so we follow the lead of Rushby, 1988, in suggesting that KBS requirements are more testable if they include competency

requirements, service requirements, and implementation requirements. (Rushby mentioned the first two.) "Functional" requirements, a possibly more familiar term, are here split between competency and service.

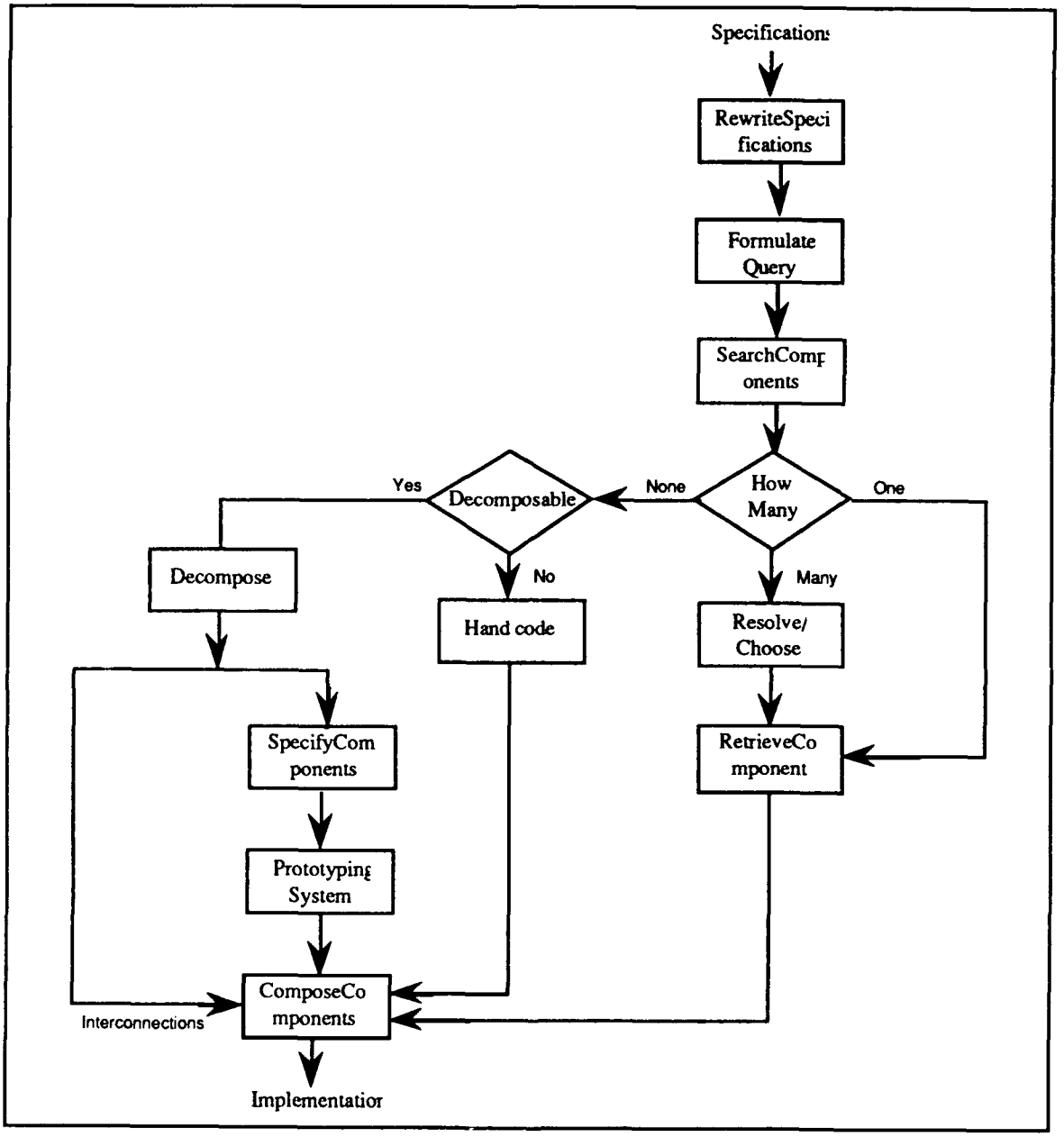


Figure 5.2—Prototyping with component library

Competency requirements may be difficult to establish, but service requirements are likely to be implicit in design of the system of which the specific component is a part. Typically, a prototype will define some of the *requirements* for service and possibly implementation, and will give *design* guidelines for dealing with competency issues. Sometimes the design guidelines resulting from a prototype are negative: don't do such-and-such the way the prototype does it because it's too slow/doesn't work in a certain case/etc.

Requirements vocabulary:

- **Generic application(s) or task(s)**
- **Prototype**

Requirements categories:

- **Competency requirements**
- **Service requirements**
- **Safety requirements**
- **Implementation requirements**

Competency requirements can be given as (1) desired competency and (2) minimum acceptable competency, again following Rushby, 1988. Minimum acceptable competency should be stated in such a way that test cases can be developed with no misunderstanding. Ideally, a set of specific test cases will be provided and the desired and minimum competency can be stated as the percentage of solutions that are correct or within limits. Where a reasonable set of specific test cases cannot be identified because of an overly-large or unknown problem space, a description of the likely *state space*, giving definite limits where possible, should be provided instead.

Service requirements include response time, availability, capacity or throughput, interfaces, formats, and so forth. This is similar to the non-functional requirements of most procedural systems. Reliability can be considered a subset of service, and reliability requirements may take the form of number of failures per unit service time. "Failure" needs to be strictly defined, probably as not meeting one of the other—service or competency—requirements.

Safety requirements could be considered a separate category, or may be placed within service requirements. In either case, safety requirements include things a system shall *not* do, complementing the other requirements that specify what a system shall do.

Implementation requirements are less common and are sometimes gratuitous. In the case of KBSs they include hardware platforms, languages, heuristics and algorithms in cases where there is reason to constrain developers' choices.

The form that requirements take is important for the later validation. For service requirements, methods appropriated from the existing world of procedural systems are appropriate. Boehm, 1984, reviews some special languages for the unambiguous statement of requirements.

For competency requirements, however, two forms of statement should be considered. First, explicit criterion test cases should be defined, with the limits of "acceptable" and "desired" responses. These cases should exercise the "normal" range of system functions as well as significant boundary conditions that a tester not intimately acquainted with the domain might not come up with. Second, for more difficult problems where at the time of writing requirements there is no ready consensus on correctness, another set of test cases should be defined, where the system's performance will need to be judged by a panel of experts; the criteria for selection of panel members should be stated in the requirements.

5.5.3 Knowledge Base Development Methodology

Most existing methodologies, such as MIL-STD 2167, stress technical details and traceability from one development stage to the next, resulting in verifiable systems. Validation, however, is less well covered because the initial steps of requirement specification and the high-level translation to design concepts are relatively open-ended and not constrained to use particular concepts. Furthermore, existing procedures contain the potential for poor designs resulting from an over-emphasis on traceability. This occurs where developers specify an implementation module for each requirements function, where this is not an optimal implementation breakdown, in order to ease the traceability task

The methodology discussed here for ALS lends itself much more to system validation. Rather than imposing requirements inappropriately on implementation, the emphasis is on providing a range of generic implementation modules from which achievable requirements can easily be specified. This serves the purposes discussed earlier of lowering the level of abstraction of the requirements language and raising the implementation language. An analogous but less efficient technique is using high-level tools and prototyping to determine requirements middle-out (and to assist making design decisions): a prototype makes an excellent statement of requirements for a computer program because (a) it is a computer program and therefore has features corresponding to the target, and (b) it clearly can be implemented. However, a system developed from a prototype still has to face the question of what are the real requirements.

The proposed ALS methodology is also appropriate for procedural or knowledge-based systems. Specific procedures and details such as documentation formats are not described; these need to be tailored to specific development environments and applications.

In the suggested knowledge base development methodology there is no sharp distinction between initial prototypes to gather requirements, later prototypes to explore designs, and final delivery systems. The same principles apply in each case, although it is obvious that the priorities differ. When trying to establish requirements, the details of knowledge base internals are probably of no concern at all, but they become more significant as the concepts and designs are finalized. See Miller, 1989, for an example V&V methodology that includes this dimension.

5.5.3.1 Overview

The development methodology requirements follow the guidelines developed in the discussion above concerning V&V sensitivities. They are tailored for an environment that supports distributed KBSs, including such functions as distributed DBMS, knowledge transfer between cooperating but independent systems, and coordination functions like blackboards. (See Section 4, K-bus, for an example of such an environment.) Without such an environment, mechanisms for cooperation between subsystems must be built from scratch each time, the complexity of the overall system grows exponentially as components are added, nothing is standard, and we are no better off than today where making a system the size of ALS (nearly-) completely autonomous would be impossible.

On top of and integrated with this environment that supports distributed KBSs, should be developed a library of generic KBS tasks, such as those proposed by Chandrasekaran, and, at a higher level yet, generic applications. While applications are developed top-down by identifying which generic applications, tasks, and heuristics they exemplify, the support library is developed bottom-up, from support environment to general heuristics and functions to generic tasks to generic applications.

> **Top-down development from a library of verified compo–nents is recommended, bearing in mind that prototyping is an excellent way to explore and evaluate alternative requirements and designs.**

Within this framework, a top-down development effort is advocated, from requirements definition (see above) to knowledge acquisition or design, development, check-out and testing. Where a problem is not sufficiently understood for top-down development, prototyping *at each stage* (not just for requirements) can be useful for evaluating alternatives and making informed decisions. The specification at each stage should be captured in a form that will permit automated verification tools to compare it with the products of the next stage. (Documentation/artifact tools and standards, and verification tools, should be developed together to enable this degree of automation.)

5.5.3.2 Knowledge Acquisition and Documentation

Using the hierarchical architecture of generic application-generic task-generic heuristic from the beginning rather than as an after-the-fact documentation concept, knowledge acquisition should strive to define problems as specializations of generic applications and tasks. A statement of the form, "first we do this,

then this, then this,..." may be a valuable first step, but the knowledge has not been acquired until it can be characterized in terms of one of the generic modules plus certain unique properties or listed specifically as an exception. A database of such characterizations can be used to identify common exceptions to the generic applications/tasks/heuristics library and plan further additions to the library by generalizing one-of-a-kind solutions.

Where human expertise seems to be the only way to deal with a requirement, the human should consider himself a requirements gatherer and knowledge engineer as much as an in-the-loop operator. By prototyping and developing incrementally, a set of computerized tools to assist the human can be developed; instrumenting the tools to track how they are used can lead to extra knowledge of solution strategies even where it seems that everything is *ad hoc*.

5.5.3.3 Design Guidelines

Because ALS is being computerized from the ground up, we assume that it will be possible to design top-down, starting with a system-wide end-to-end design. From the high-level design should come hooks for computers at appropriate places in all subsystems, and many built-in data sources allowing data to be captured as soon as it becomes available, leading to data efficiencies and potential re-use. Because of the scope of ALS, it will not be feasible to carry such an end-to-end design completely down to the level of individual knowledge bases and programs initially, so a hierarchical decomposition will have to be adopted, where high-level tasks are progressively subdivided until feasible modules are arrived at. The KBS development methodology should itself be inherently hierarchical to some extent to map neatly into the overall system.

At a more detailed level the KBS development methodology's design guidelines should emphasize three objectives: design for safety (see Section 5.5.3.7, below), design for maintenance and design for reusability. (Obviously, to design for correct problem-solving, meeting performance requirements, etc. is assumed.)

> **KBS design should incorporate defensive measures to ensure safety, check limits, and confine side-effects.**

At a more detailed design level, individual KBSs (and procedural programs) should strive to implement run-time verification by doing some of the following:

- display status if requested even if running autonomously
- accept human command to stop safely even if running autonomously
- halt when confronted with unknown territory
- include legal range of values in variable declarations (and abstract sensor definitions)
- recognize and do something useful when out-of-range data encountered (based on the description of the system's legal state space, included in the requirements)
- constrain sensors in terms of the data values they are permitted to produce
- post overall status on a blackboard (see Section 4); based on this high-level status, probe commands for potentially hazardous events
- block hazards if a problem is observed

One approach might be to modify implementation languages and corresponding inference engines to work like Ada in terms of permitting only certain data values. Alternatively, an object-oriented language such as C++ is a supportive environment for this approach.

Where critical functions are involved, a degree of run-time testing may be built into knowledge bases by redundant design, whereby either multiple solutions are generated, or a single solution is generated but then evaluated independently. According to Linden, 1988:

- It is often easy to develop independent solutions as long as they are not required to be as "good" as the principal one: a "back of an envelope" check may suffice. In this form of run-time testing the check solutions are used only to assess the reasonableness of the principal one.
- In planning and design problems it is relatively easy to validate the *adequacy* of a solution, even if determining optimality is hard.

5.5.3.4 Module Structure

Object-oriented concepts should be used in the determination of modules within knowledge bases and their corresponding documentation standards. A module in this sense is a body of code that is configuration managed and documented as a unit, such as a file or collection of files.

Modularized design is the objective.

- modules should be designed for hierarchical assembly as outlined above with interfaces generally "up" and "down," not "sideways" (in the hierarchy sense), although violating this guideline may not be serious if object-oriented design and implementation techniques are followed
- all rules and facts relating to a domain (e.g., liquid oxygen, seawater quality, etc.) should be together in one module, even if their use is scattered throughout an application
- there should be minimal external interfaces to a module
- external interfaces should be explicit rather than through global data
  - input may be through global data
  - output should not change global data that "belongs to" higher-level modules
  - output data may be global to lower-level subproblems
- eliminate/minimize side-effects among domains

Specific versions of these guidelines and tools to check for adherence to them need to be provided for each design and implementation language.

5.5.3.5 Static Analysis of Heuristics and Algorithms

Automated static analysis of knowledge base structure is demonstrably doable and can be relatively complete and extensible.

Automated knowledge base analysis tools like EVA can be a valuable part of V&V.

The Expert System Validation Assistant (EVA), described by Stachowitz, 1989, is a good example of such a tool. Another is the CRSV module in CLIPS (Culbert and Savely, 1989). (See also Laurent and Ayel, 1989, Kiper, 1989, and Landauer, 1989.) Currently EVA has the following functions:

- translation between ART, KEE and LES knowledge base formats and EVA's standard format; other translations are feasible (e.g., OPS83, CLIPS)
- [extended] structure checker: contains intelligent syntax checking; detection of apparently superfluous rules, conditions, facts and objects; detection of circularity, unreachable facts, objects, dead-ends
- [extended] logic checker: finds inconsistent and conflicting rules
- semantics checker: based on a user-supplied metaknowledge base describing constraints in the knowledge base, finds constraint violations
- omission checker: builds logical decision tables from rules and identifies situations not covered
- rule refiner: given test cases with known results, check if the outcomes match what is expected; where outcomes do not match, refine the rules (used for diagnostic/classification systems)
- control checker: check control structures (e.g., IF-THEN-ELSE) for completeness; determine whether developer-specified sequencing constraints are met
- test case generator: help the developer generate appropriate test cases derived from the system's understanding of the system and its limits
- error locator: locate rules that derive incorrect facts from input facts
- behavior verifier: derive a description of the system's behavior; compare this with a specification of the intended behavior

Figure 5.3 illustrates where these kinds of functions fit in the knowledge acquisition and testing process.

For procedural modules, some existing mathematical proof-based techniques exist, but Fetzer, 1988, has argued that the very concept of automatic program verification is not reasonable. He contends that the provable part of a program, the core algorithms, is very small in relation to all the unverifiable code concerned with getting user inputs, opening files, and so forth, and that it is this peripheral code that causes most of the errors in large systems.

In another vein, existing verification methods like structured walk-throughs are easily adapted to KBS, where they are especially relevant. The multi-viewpoint environment of a walk-through is valuable when dealing with systems where subtle interactions and heuristic approaches are involved. Other comprehension aids could also be produced automatically, for example a graphic output addition to EVA.

As well as relatively objective attributes such as those EVA is concerned with, engineers recognize that a "good" design has certain subjective qualities. Hayes-Roth, 1989, nominates some intrinsic engineering qualities of KBSs, including elegance of design, modularity, architectural quality (simplicity of interfaces), prototyping support, graphics support and database support. As advances in knowledge engineering are made it may become feasible to measure these in some meaningful way and the development methodology and its verification can be extended accordingly.

When using automated tools to verify knowledge bases, one must bear in mind their limitations. The tools can describe a knowledge base in terms of specific structural or statistical models, but cannot know the "real" problem the knowledge base is designed to solve, the assumptions underlying its creation, nor therefore any of its essential semantics. In the words of one observer:

> "A program given no semantic information cannot hope to determine the validity of a model expressed as rules. It can only indicate to an expert where strange things happen, so that the expert can then decide if they are indeed strange things in the application or just strange things in the rules." (Landauer, 1989)

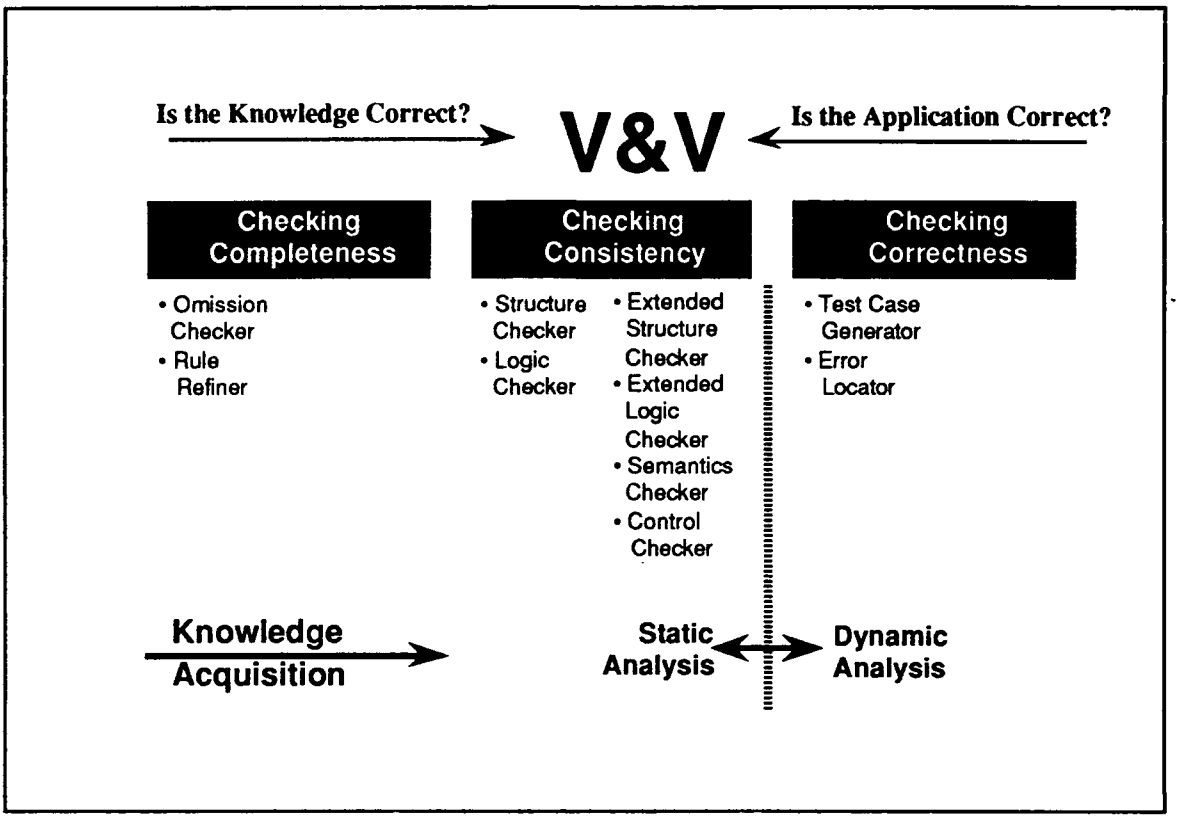


Figure 5.3—Role of V&V Associate during development

5.5.3.6 Automatic Rule Induction

Automated rule induction techniques, originally proposed as a means of overcoming difficulties in knowledge acquisition ("the bottleneck problem of applied artificial intelligence" —Feigenbaum, 1977), are automated procedures for deriving rulesets from large numbers of examples (Michie, 1987). The expert's job is to select representative training examples (input data and output classifications), which are presented to a system running an algorithm such as ID3 (Quinlan, 1979). The system produces rules that duplicate the input classifications consistently, as well as analyses of the quality and variance of the classifications. For example, Michie, 1987, describes use of a program called ExTran 7 by Rocketdyne to analyze and classify voluminous (approx. 50 megabytes per test) digital test data obtained from rocket engine firings. Burke and McNenny, 1989, argue that ID3 is very effective way to develop classification systems.

A couple of possible benefits follow from the way that the rules are derived automatically by an algorithm. First, an extremely large volume of example data may be considered; and second, the application of the algorithm will be consistent by definition and therefore verified to produce a certain level of correctness in advance. Further, the resulting rules may be further refined manually to improve desired aspects of performance (Quinlan, 1987, gives an example of optimizing by pruning induced rulesets).

However, there is a problem with this technique, resulting from its essential mindlessness. Although the rules that are created may efficiently cover the training data, they are not derived from a consideration of the essential underlying principles and may misconstrue basic processes (without realizing it) and not scale up to different data. Thus, it is essential that the training database be comprehensive and from a known population. Possibly the technique should be limited to classification problems where there is a large body of historical data and no hazards and the problems presented can be constrained so as to come from the same population as the training set.

Automated rule induction is a useful—and "pre-verified"— way to explore problem spaces, but knowledge engineers should carefully review and edit the rules so produced.

Perhaps the way to view the technique is as a way of exploring a problem domain and deriving an initial tentative set of rules. These rules and the training cases that went into them may then be analyzed for trends that were not evident at a more theoretical level. If the rules seem to have general validity, they may be used, at least in part, with modifications.

5.5.3.7 Dynamic Sensitivity Analysis

Verification of robustness or lack of sensitivity is generally a determination that outputs are regularly related to inputs over the whole spectrum of potential inputs, and that there are no unexpected discontinuities, such as at the edge of a system's domain of competence. Integration sensitivity refers to some systems' tendency to become unstable or to change their apparent behavior when integrated with certain other components, compared to their operation in isolation. Although some sensitivity can be identified through static analysis (such as identifying where the "edge of competence" is), dynamic testing is important.

Ideally, the tools to perform dynamic testing will be built-in, in the knowledge base or inference engine: "demon" rules that specify the boundary of competence and signal all crossings of it, and built-in analysis tools. For instance, CRSV, a part of CLIPS, has been extended to provide a run-time trace describing the execution of the knowledge base. This can be examined to identify out-of-competence conditions, for example, input data that is so unexpected that no rule has been provided to deal with it and it is simply ignored. The developers of CLIPS are examining ways to integrate this sort of analysis more fully into the run-time environment. (Culbert and Savely, 1989)

5.5.3.8 Verification of Safety

A formal hazard analysis to determine exactly which sorts of failures can propagate to hazards will probably be required at a higher level for many ALS systems. Such an analysis identifies the most critical failures; where they can be caused by software, special design steps must be taken for their prevention.

At a general level, a recommended approach is the well-known one of designing so that all potential hazards are blocked unless explicitly enabled. In KBSs there is a potential problem with this because the high priority accorded safety may not map into a high priority in a program's problem space at any given moment. A recommended solution is to use the hierarchical architecture, and separate the problems of status determination (is everything safe to perform action *x*?) from the general problems a system is trying to solve, and to locate the safety status determination logically close to the relevant effectors. Thus, a separate KBS, that may be termed a probe of the problem space or a demon, can be assigned the single task of determining when all criteria are met to allow the activation of a potentially hazardous effector; the other KBSs go about their business and set the values of the criteria as by-products of their work. Verification can then focus on the safety of these KBSs' knowledge bases without having to untangle them from the larger problems they are part of. One approach might be to have such control strategies built right into sensors and effectors.

5.5.3.9 Reusability

If software components can be reused, the V&V that has been invested in them can be preserved. Although a system built of certified components must itself be verified and validated, verification of the specific tasks carried out by the components can be assumed to the extent that the components have no side-effects or extraneous interfaces and the host system does not induce misbehavior.

To prepare for reuse, the conditions under which a KBS module is known to be correct should be part of each component's certification. For example, a knowledge base module that analyzes circuits can be certified to have only specific limited interfaces and certified to work correctly when run under a (certified)

CLIPS or ART inference engine, if the circuit diagram description knowledge is derived from a certain (certified) CAD system by a specific (certified) utility program.

Documentation standards promote reusability, avoid re-invention of the wheel.

To move in this direction, the design and verification documentation for knowledge base modules (and procedural modules) should state the following:

- algorithm or heuristic it implements, if any
- generic task it implements, if any
- generic application it implements, if any
- specific application it implements, if any
- ALS-wide domain knowledge it incorporates , if any
- application-specific domain knowledge it implements, if any
- testing it has undergone and expected results (embed regression test?) at the level of algorithms, tasks and/or applications

All of the "generic" elements should be available in a central library or database (Nicoud, 1988; Goodwin and Fussell, 1989, use the term *electronic knowledge repository*), i.e., in Unis, and all of the applicable descriptors should be available in an index database. Such a library would generate some side benefits. For instance, validity checking tools (see the discussion of EVA, 5.5.3.5, above, for an example) could be designed to check the library as well as individual modules, perhaps resulting in a large degree of global inter-module consistency and compatibility, and maximum data sharing through the detection and elimination of redundancy. Additionally, the central library of knowledge could be represented in a single abstract knowledge representation language and automatically translated (in a presumably error-free manner) to the languages of target inference engines (this is also a feature of EVA, see above).

5.5.3.10 Real-time Performance Analysis

The real-time issue is raised with KBSs because their data-driven character makes them relatively unpredictable performers, dependent upon their current problem space. A KBS can be made *fast* (soft real-time), but not necessarily *consistent* (hard real-time), and consistency is more important when designing a real-time system. ALS has the additional complication of being a distributed system, which creates both obstacles (communications overhead) and potential solutions (parallel execution) to performance problems.

Real-time KBSs are difficult, but there are design ap–proaches that can make them run fast. Developers should consider alternative solutions where critical real-time performance is required.

Various approaches should be adopted:

- minimize KBS problem spaces through modular design so individual KBS that is expected to exhibit high performance is not loaded down with irrelevant data
- benchmark inference engine performance in particular tasks and select an appropriate one. For example, in forward-chaining rule-based systems OPS83 is significantly faster than CLIPS or any of its other logical equivalents; most LISP-based inference engines are relatively slow; PROLOG implementations differ widely in performance,

but can be very fast. Similarly, different knowledge base structures can result in different performance levels with a given inference engine.

- minimize message traffic, pass only high-level status information, not individual bytes of data
- restrict data and message traffic at low levels rather than clogging up KBSs' problem spaces. For example, if a data rate increases substantially, the sensors can reduce their sensitivity so they continue to send a similar number of data transactions to a KBS
- use real-time operating systems at certain nodes as long as network interfaces (e.g., TCP/IP) can be preserved. Then, implement the most real-time-critical operations procedurally under this real-time OS on a priority interrrupt basis rather than through a KBS
- select an implementation that compiles out layers in a hierarchical architecture. For example, an object-oriented language like C++ integrates the hierarchy through inheritance, which is resolved at compile time, rather than through run-time procedure calls.
- implement distributed systems with an architecture that supports plug-compatible component replacement, to allow selective replacement of components with faster ones

Design guidelines will have to be prepared along these lines and will become inputs to the verification process. If some of the suggested steps are not taken and KBS are required to give real-time performance with unlimited potential problem spaces, then validation will be practically impossible.

5.5.3.11 Tools

It is possible to provide specialized tools to assist in performing stage-to-stage transformations, and methodologies for procedural programs show several examples, such as PSL/PSA, SREM, Software Through Pictures, and a number of module design tools (see Boehm, 1984, for an overview and description of SREM). These tools also can review specifications for correctness at least at the level of syntax and traceability, if not semantics, and can produce required documentation. These features can be valuable, especially traceability aids that ensure that all required interfaces are dealt with, and automated documentation producers. There is no reason that, at least down to the module-to-module interface level, such tools could not also be used for KBSs.

Within a knowledge base, however, KBSs are different enough from procedural programs' code that different tools, along the lines of EVA, would be required.

Tools to automate testing are widely used with procedural programs (see Myers, 1979, for an extensive review and bibliography), and the concepts have application also to KBSs. Execution and configuration management tools that perform and track tests and maintain libraries of regression tests are immediately applicable. However, static analyzers, test planners, test coverage analyzers, symbolic executors, and so forth will not port from the procedural to the knowledge-based worlds, and new tools, again along the lines of EVA, will be needed.

5.5.3.12 Certified Inference Engines

One class of KBS tools that can be overlooked is the inference engines or "shells" employed to execute knowledge bases. The languages or paradigms the inference engines execute need to be completely defined in syntax and semantics, including all boundary conditions such as null, missing, maximum and minimum values. As well, the inference engine language definitions must include the elements of procedural languages as appropriate, including I/O, data types, math operations, and all the procedural functions available in the language. Then, finally, the inference engines need to be certified against their definition in the way that DoD certifies an Ada environment or a FORTRAN compiler.

Verification requires certified inference engines, just like compilers for conventional languages.

Any interoperative libraries supporting an inference engine, such distributed system components should be similarly certified. This goes some way to meeting the objective of having totally certified components so that *verifying* that a program is correctly constructed of these components goes a long way towards *validating* the program.

5.5.3.13 Configuration Management

Carrying out effective V&V—whether a design review, knowledge base consistency check or validation test—and post-release maintenance requires precise knowledge of just what software is being verified or executed. A configuration management program is therefore an essential complement to V&V. Relatively automated and flexible CM is especially important in a prototyping mode, where a system and its definition at all levels from broad requirements to fine details of code may be changing rapidly.

It goes without saying that where a library of generic applications and tasks (or "electronic knowledge repository") is used as the basis of development, the library itself must be under strict configuration control. For modules to be entered into the library it should be proved that a sufficient level of documentation is provided for the benefit of both users and later maintainers, and that each module has itself undergone a critical V&V process.

5.5.3.14 Evaluation of V&V

Any on-going function within development should be continually monitored for effectiveness. The ALS V&V tools should be designed to record the results of their activities (as part of configuration management) so these tools and the procedures they implement can be evaluated and modified as system performance data becomes available. Dunham (1989) proposes four types of evaluations: descriptive evaluations, case and multiple-case studies, formal experiments and quasi-experiments, the first two of which are most appropriate to ALS.

5.5.4 Testing

Testing is the principal means to validate a KBS, and is not as closely tied to the development methodology as verification. Following the breakdown of KBS requirements, testing can be broken down into testing against competency requirements and service requirements.

5.5.4.1 Competency Testing

Competency testing of a KBS can be difficult where there is a huge potential problem space, and when the definition of competency itself is not a simple binary choice (and it will frequently not be). However, if some of the design guidelines suggested above have been followed and a modular approach followed, then each KBS component should be dealing only with relatively well-defined problems and should be in receipt only of limited data. Another cause for difficulty with competency testing, lack of competency requirements, should not apply to ALS.

As stated above, requirements should include a set of test cases defining critical aspects of minimum and desired competency. Testers may also want to develop their own working definition of the levels of competency in terms of the implementation strategy (rather than the requirements) to help develop additional cases. O'Keefe, Balci and Smith, 1987, discuss the concept *acceptable performance range*, arguing that it is likely to be quite application-specific. For instance, it may be vital that the system never make a specific error with serious consequences, such as classifying an A as a B incorrectly, although a certain percentage of other misclassifications may be acceptable.

Overall, the approach to testing should be structured, with the objective being to find errors rather than to show that the system works correctly. Structure in this sense means to determine the range of situations a program will encounter, and systematically cover that range; if the program blocks portions of the range,

attempt to defeat the blocking. A fair means to this end that has some inferential validity is random selection of test cases using stratified sampling—that is, randomly sampled within each problem space type. For example, if a KBS may classify cases as A, B, or C, and experience shows that these cases occur 70%, 10%, and 20% of the time, respectively, then the test cases should be in the same proportion. (O'Keefe, Balci and Smith, 1987). Fairness should not preclude completeness, however, and sufficient extra "hard" cases should be included to ensure coverage at the margins.

Regression testing is always important, and involves saving previous test cases under configuration control and re-running them after changes. Automated test conductors are important to assuring complete regression test coverage.

As well as their use in regression tests, previous test cases should be saved for testing other systems with analogous requirements and functions, thus saving some of the investment in deriving the tests in the first place.

An important set of tests is to confirm the correctness of any design approaches adopted to limit the potential problem spaces. This can be achieved by a form of white box testing where the operation of internal mechanisms in a KBS designed to filter inputs and restrict the system to a certain problem domain are observed. EVA provides some of this with the semantics checker, but an alternative and perhaps more efficient approach is to implement filtering at the sources of data, whether other subsystems, blackboards, or sensors, rather than within the KBS itself.

The extent of competency testing required should be related to the costs of making a validation error. Type I errors (accepting an invalid system) will cost the users at a later time, while type II errors (rejecting a valid system) cost the developers. Steps to prevent one type of error tend to increase the likelihood of the other. In autonomous systems, type I errors can be very costly, and prevention of type I errors typically requires more stringent testing or a higher standard of competency. Where a system is not autonomous, with human back-ups on hand, type I errors may be acceptable.

5.5.4.2 Service Testing

Service requirements, dealing with interfaces, response time, formats, etc., are very similar to the requirements for procedural software, and there exists a large body of experience and literature (e.g., Myers, 1979, and the references therein) for guidance on testing for these requirements.

5.5.4.3 Operations and Maintenance

In the "waterfall" software development model, maintenance is defined as a separate phase that follows and is somehow divorced from development. In practice, however, there is little distinction between a cycle of iteration during initial development and a cycle of maintenance, and V&V procedures should continue in force.

Operations after delivery can be thought of as extended testing and a continuation of iterative development. Procedures for recording apparent errors and bringing them to the attention of development/maintenance personnel, as occurs with most procedural programs today, need to be in place for all deployed KBSs. The development/maintenance personnel will need access to the knowledge base library and documentation that explains the operations and underlying assumptions of the various modules.

5.6 REFERENCES CITED

Alford, Mack. "SREM at the age of eight: the distributed computing design system." *IEEE Computer*, April 1985.

Barrett, Britt W. "Software reliability models: can they be applied to expert systems?" IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Boehm, Barry W. "Verifying and validating software requirements and design specifications." *IEEE Software*, January 1984.

Boehm, Barry W. "A spiral model of software development and enhancement." *IEEE Computer*, May 1988.

Bundy, Alan. "How to improve the reliability of expert systems." *Research and development in expert systems IV: proceedings of Expert Systems '87* (British Computer Society), Cambridge University Press, 1988.

Burke, Roger A., and Robert W. McNenny. "VV&T advantages provided by the ID3 algorithm." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Cha, Stephen, Nancy Leveson and Timothy Shimeall. "Safety verification in Murphy using fault-tree analysis." IEEE something, (?source?) 1988.

Chandrasekaran, B. "Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design." *IEEE Expert*, Fall 1986.

Citrenbaum, R.L., and J.R. Geissman. "A practical cost-conscious expert system development methodology." AI-86 Artificial intelligence and advanced computer technology conference, April 1986.

Culbert, Chris, and Robert T. Savely. "CRSV and AAMP: Ongoing work at NASA/Johnson Space Center in KBS verification and validation." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

DoD-STD-2167A. Defense Systems Software Development. February, 1988.

Duke, Eugene L. "V&V of flight and mission-critical software." *IEEE Software*, May, 1989.

Dunham, Janet R. "V&V in the next decade." *IEEE Software*, May, 1989.

Fetzer, James H. "Program verification: the very idea." *Communications of the ACM*, September, 1988.

Feigenbaum, E.A. *The art of artificial intelligence 1: themes and case studies of knowledge engineering.* Department of Computer Science, Stanford University, 1977. (Quoted in Michie, 1987.)

Geissman, James R. and Roger D. Schultz. "Verification and validation of expert systems." *AI Expert*, February, 1988.

Goodwin, Mary Ann, and Charles C. Robertson. "A systemic view of validation and testing knowledge-based systems." Validation and testing knowledge-based systems workshop, AAAI conference, 1988.

Goodwin, Mary Ann, and Louis R. Fussell. "Use of electronic knowledge repositories in VV&T of knowledge-based systems." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Gruman, Galen. "Major changes in federal software policy urged." *IEEE Software*, November, 1989. (Summary of "Bugs in the program," a report prepared for the House Committee on Science, Space and Technology's Subcommittee on Investigation and Oversight, US Government Printer, October, 1989.)

Hayes-Roth, Frederick. "Towards benchmarks for knowledge systems and their implications for data engineering." *IEEE Transactions on knowledge and data engineering*, March 1989.

IEEE Standard for software verification and validation plans, American national standard ANSI/IEEE Std 1012-1986, IEEE, 1986.

Johnson, Sally C. "Validation of highly reliable, real-time knowledge-based systems." SOAR 88 workshop on automation and robotics, Dayton, OH, July 20-23, 1988.

Kiper, James D. "Structural testing of rule-based expert systems." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Landauer, Christopher. "Principles of rulebase correctness." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Laurent, J.P., and M. Ayel. "Off-line coherence checking for knowledge base systems." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Leveson, Nancy G. "Software safety: why, what, and how." *Computing surveys*, vol. 18, no. 2, June, 1986.

Leveson, Nancy G. "Safety as software quality." *IEEE Software*, May, 1989.

Linden, Theodore A. "Alternative approaches to V&V for AI systems." Validation and testing knowledge-based systems workshop, AAAI conference, 1988.

Linden, Theodore A. "A meta-level software development model that supports V&V for AI software." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Luqi. "Software evolution through rapid prototyping." *Computer*, May, 1989.

Luqi, and Valdis Berzins. "Rapidly prototyping real-time systems." *IEEE Software*, September, 1988.

Luqi, and Mohammad Ketabchi. "A computer-aided prototyping system." *IEEE Software*, March, 1988.

McCulloch, W.S. and W.H. Pitts. "A logical calculus of the ideas immanent in nervous activity." *Bulletin of mathematical biophysics*, vol. 5, 1943.

Michie, Donald. "Current developments in expert systems." In J.R. Quinlan (ed.), *Applications of expert systems*, Addison-Wesley, 1987.

Miller, Lance A. "A comprehensive approach to the verification and validation to knowledge-based systems." IJCAI-89 workshop on verification, validation and testing of knowledge-based systems, August 1989.

Myers, Glenford J. *The art of software testing*. Wiley-Interscience, 1979.

Nicoud, Stephen. "Requirements for a database system to support development of large-scale expert systems." Validation and testing knowledge-based systems workshop, AAAI conference, 1988.

O'Keefe, Robert M., Osman Balci, and Eric P. Smith. "Validating expert system performance." *IEEE Expert*, Winter 1987.

O'Leary, Daniel, and Robert O'Keefe. "Verifying and Validating Expert Systems." Tutorial presented at the Eleventh joint conference on artificial intelligence (IJCAI-89), 1989.

Quinlan, J. R. "Discovering rules by induction from large collections of examples." In D. Michie (ed.), *Expert systems in the micro electronic age*, Edinburgh University Press, 1979.

Quinlan, J.R. *et. al.* "Inductive knowledge acquisition: a case study." In J.R. Quinlan (ed.), *Applications of expert systems*, Addison-Wesley, 1987.

Ross, Douglas T. "Applications and extensions of SADT." *IEEE Computer*, April 1985.

Rushby, John. *Quality measures and assurance for AI software*. NASA contractor report 4187 (SRI International), October 1988.

Stachowitz, Rolf. "Verification and validation of knowledge-based systems." SOAR 89 workshop on automation and robotics, NASA JSC, Houston, TX, July 25-27, 1989.

von Neumann, John. "Probabilistic logics and the synthesis of reliable organisms from unreliable components." in *Automata studies*, C.E. Shannon and J. McCarthy (eds.), Princeton University Press, 1956.

Wallace, Dolores R., and Roger U. Fujii. "Software verification and validation: an overview." *IEEE Software*, May, 1989.

Winograd, S., and J.D. Cowan. *Reliable computation in the presence of noise*. MIT Press, 1963.

6. KNOWLEDGE BUS BIBLIOGRAPHY

The concepts and approaches for the K-bus have not been carried out in isolation. Instead they borrow heavily from the leading work in the areas of real time systems, distributed systems, distributed artificial intelligence, object oriented design and programming, and the core artificial intelligence areas, such as knowledge representation. The bibliography that follows contains those papers we deemed most relevant to the K-bus concepts.

The citations include the normal bibliographic information, plus the identity of the related system, if there was one, the underlying language used, the institution doing the work, and the funding institution, when known.

ABRA82

author: Abramovici,M. Breuer,M.

title: **Fault Diagnosis in Synchronous Sequential Circuits Based on an Effect-Cause Analysis**

source: IEEE Transcations on Computers, Vol 31, No. 12, December 1982

categories: Fault Diagnosis

system: *language:* 8 *Pages*

institution: Bell Labs & USC *funded by:* NSF

ADLE89

author: Adler, R.M., Astrid Heard, R. B. Hasken

title: **OPERA - An Expert Operations Analyst for a Distributed Computer System**

source: Proceedings AI in Goverment Conference, March, 1989

categories: Real Time

system: OPERA *language:* 8 *Pages*

institution: Bell Labs & USC *funded by:* NSF

AGHA86

author: Agha,G. Hewitt,C.

title: **Concurrent Programming Using Actors: Exploiting Large-Scale Parallelism**

source: Bond-Gasser

categories: Distributed Systems, Distributed AI, Knowledge Representation

system: ACTORS *language:* Act3 10 *Pages*

institution: MIT *funded by:* System Development Foundation

AKSI87

author: Askit,M.

title: **Distributed Operating Systems, An Overview**

source: NTIS

categories: Distributed Systems

system: *language:* 23 *Pages*

institution: University of Twente, Netherlands *funded by:*

ALI85

author: Ali,M. Scharnhorst,D.

title: **Sensor-based Fault Diagnosis in a Flight Expert System**

source: CAIA-85

categories: Fault Diagnosis, Real Time Systems

system: FLES *language:* LISP 6 *Pages*

institution: University of Tennessee Space Institute *funded by:* FAA & NASA-Langley

ALI86

author: Ali,M. Scharnhost,A. Ai,C. Ferber,H.

title: **A Flight Expert System (FLES) for On-Board Fault Monitoring and Diagnosis**

source: SPIE Vol 635 (Applications III -1986)

categories: Fault Diagnosis

system: FLES *language:* LISP 4 *Pages*

institution: University of Tennessee Space Institute *funded by:* FAA & NASA-Langley

ALLE83

author: Allen,J.

title: **Maintaining Knowledge about Temporal Intervals**

source: CACM, Volume 26, Number 11

categories: Temporal Reasoning

system: *language:* 12 *Pages*

institution: University of Rochester *funded by:* NSF

ALLE85

author: Allen,J. Hayes,P.

title: **A Common-Sense Theory of Time**

source: IJCAI-85

categories: Temporal Reasoning

system: *language:* 4 *Pages*

institution: University of Rochester *funded by:* ONR

AYAC82

author: Ayache,J. Courtiat,J. Diaz,M.

title: **REBUS, A Fault-Tolerant Distributed System for Industrial Real-Time Control**

source: IEEE Transactions on Computers, Vol 31 No 7, July 1982

categories: Real Time Systems, Distributed Systems, Control

system: REBUS *language:* 11 *Pages*

institution: CNRS, France *funded by:* DGRST

BADR88

author: Badrinath, B. R., and Krithi Ramamritham

title: **Synchronizing Transactions on Objects**

source: IEEE Transactions on Computers 37(5), May 1988

categories: Distributed Systems, Object Oriented

system: *language:* 7 *Pages*
institution: University of Massachusetts, Amherst *funded by:*

BAKE86

author: Baker,T. Scallon,G.
title: **An Architecture for Real-Time Software Systems**
source: IEEE Software, May 1986
categories: Real Time Systems
system: REX *language:* 9 *Pages*
institution: Florida State Univ & Boeing *funded by:*

BALZ87

author: Balzer, Robert M.
title: **Living in the Next Generation Operating System**
source: IEEE Software, November 1987
categories: Distributed Systems
system: AP5+GIST base *language:* LISP 8 *Pages*
institution: USC ISI *funded by:* DARPA

BARA88

author: Barachini,F. Theuretzbacher,N.
title: **The Challenge of Real-time Process Control for Production Systems**
source: AAAI-88
categories: Real Time Systems
system: PAMELA *language:* 5 *Pages*
institution: ALCATEL-Austria *funded by:* ALCATEL

BATE86

author: Bates,W. Holliman,C.
title: **MCC Flight Operations Automation of STS Orbiter Systems Monitoring**
source: ??
categories: Distributed AI
system: *language:* 25 *Pages*
institution: NASA-JSC *funded by:* NASA

BERN87

author: Bernus,P. Letray,Z.
title: **Intelligent Systems Interconnection: What Should Come After Open Systems Interconnection?**
source: NTIS
categories: Distributed AI, Distributed Systems, Knowledge Representation
system: ISI *language:* 12 *Pages*
institution: Center for Mathematics and Computer Science, Holland *funded by:*

BIRR82

author: Birrell, Andrew D., Roy Levin, Roger M. Needham, and Michael D. Schroeder
title: **Grapevine: An Exercise in Distributed Computing**
source: CACM 25(4), April 1982
categories: Distributed Systems
system: Grapevine *language:* 15 *Pages*
institution: Xerox PARC *funded by:* Xerox

BISI85

author: Bisiani, R., F. Alleva, A. Forin, R. Lerner, M. Bauer
title: **The Architecture of the Agora Environment**
source: Distributed Artificial Intelligence, Michael N. Huhns, Ed., Morgan Kaufman, 1987.
categories: Distributed AI
system: Agora *language:* CFrames 19 *Pages*
institution: Carnegie-Mellon *funded by:* DARPA

BLAC83

author: Black, Andrew P.
title: **An Asymmetric Stream Communication System**
source:
categories: Distributed Systems
system: Eden *language:* 7 *Pages*
institution: Univ. of Washington *funded by:*

BOLK88

author: Blokland,W. Sztipanovits,J.
title: **Knowledge-based Approach to Reconfigurable Control Systems**
source: American Control Conference, 1988
categories: Real Time Systems, Distributed AI, Control
system: Multigrpah *language:* LISP 6 *Pages*
institution: Vanderbilt U. *funded by:*

BONI88

author: Bonissone, P. Wood, N.
title: **Plausible Reasoning in Dynamic Problems**
source: AAAI V&V W/S 1988
categories: V & V, Simulation, Classification
system: LOTTA & RUM & RUMrunner *language:* KEEE, LISP, Ada 16 *Pages*
institution: General Electric *funded by:*

BOWE87

author: Bowen, B.
title: **Real-Time Expert Systems: A Status Report**
source: AGARD-LS-155 (NATO 1987)
categories: Real Time Systems, Control
system: *language:* 15 *Pages*

institution: Carleton U., Canada *funded by:*

BRIN88

author: Brink,J. Storey,P.

title: **Diagnostics in the Extendable Integrated Support Environment (EISE)**

source: SOAR-88

categories: Fault Diagnosis

system: *language:* KEE 7 *Pages*

institution: Battelle & Air Logistics Center *funded by:* Air Force

BURG85

author: Burg,B. Foulloy,J. Heudin,B. Zavidovique

title: **Behaviour Rule Systems for Distributed Process Control**

source: CAIA-85

categories: Distributed AI, Real Time Systems, Parallelism

system: *language:* C 6 *Pages*

institution: ETCA & others (France) *funded by:*

BURS87a

author: Bursch, P. Meisner, J. Winegar, K.

title: **A PC Based Expert Diagnostic Tool**

source: AUTOTESTCON-87

categories: Fault Diagnosis

system: PCMDS *language:* LISP & KEE 5 *Pages*

institution: Honeywell *funded by:* AFWAL

BURS87b

author: Bursch, P. Meisner, J. McAfoos, R. Schroeder, J.

title: **The Flight Control Maintenance Diagnostic System**

source: NAECON-87

categories: Fault Diagnosis

system: FCMDS *language:* LISP & KEE 6 *Pages*

institution: Honeywell & AFWAL *funded by:* AFWAL

BUSH87

author: Bush,J. Critchfield,A.

title: **The Resource Envelope as a Basis for Space Station Management System Scheduling**

source: CAISA-87

categories: Planning

system: PRESS *language:* M.1 10 *Pages*

institution: Computer Sciences Corp *funded by:* NASA-Goddard

BUTL88

author: Butler,P. Allen,J. Bouldin,D.

title: **Design and implementation of a parallel computer for expert system applications**

source: SPIE Vol 937 (Applications VI - 1988)

categories: Parallelism

system: OPSNET *language:* 8 *Pages*

institution: Oak Ridge Nat. Lab & Univ. of Tennessee *funded by:* DOE

BYLA83

author: Bylander,T Mittal,S. Chandrasekaran,B.

title: **CSRL: A Language for Expert Systems for Diagnosis**

source: IJCAI-83

categories: Fault Diagnosis, Knowledge Representation

system: CSRL *language:* LISP & FRL 4 *Pages*

institution: Ohio State University *funded by:* Battelle

CACA87

author: Cacace,R. England,B.

title: **TES - A Modular Systems Approach to Expert System Development for Real-Time Space Applications**

source: CAISA-87

categories: Real Time Systems, Fault Diagnosis

system: TES *language:* KEE & LISP 5 *Pages*

institution: United Technologies Corp *funded by:* NASA-JSC

CANT83

author: Cantone,R. Pipitone,F. Lander,W. Marrone,M.

title: **Model-Based Probabilistic Reasoning for Electronics Troubleshooting**

source: IJCAI-83

categories: Fault Diagnosis

system: IN-ATE *language:* LISP 5 *Pages*

institution: US Naval Research Laboratory *funded by:* US Navy

CARL87

author: Carlson,K.

title: **Process-Driven Expert Systems**

source: AI-87 (Long Beach)

categories: Control, Real Time Systems

system: *language:* 8 *Pages*

institution: TI *funded by:*

CAST87

author: Castillo,D. Ihrie,D. McDaniel,M. Tilley,R.

title: **An AI Approach for Scheduling Space-Station Payloads at Kennedy Space Center**

source: CAISA-87

categories: Planning
system: PHITS *language:* FLAVORS (?) 10 *Pages*
institution: Harris Corp & NASA-KSC *funded by:* NASA-KSC

CHAN86

author: Chandrasekaran,B.
title: **Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design**
source: IEEE Expert, Fall 86
categories: Knowledge Representation, Fault Diagnosis
system: *language:* 8 *Pages*
institution: Ohio State University *funded by:* Air Force Office of Scientific Research

CHAP85

author: Chapman,D.
title: **Nonlinear Planning: A Rigorous Reconstruction**
source: IJCAI-85
categories: Planning
system: TWEAK *language:* 3 *Pages*
institution: MIT *funded by:* DARPA & NSF & IBM

CHAP87

author: Chapman,D.
title: **Planning for Conjunctive Goals**
source: Artificial Intelligence Vol 32 No 3, July 1987
categories: Planning
system: TWEAK *language:* 47 *Pages*
institution: MIT *funded by:* DARPA & NSF & IBM

CHEN85

author: Chen,D.
title: **Progress in Knowledge-Based Flight Monitoring**
source: CAIA-85
categories: Control
system: SECURE *language:* LISP & FRL 6 *Pages*
institution: E-Systems *funded by:* NASA & Air Force

CORK86

author: Corkill,D. Gallagher,K. Murray,K.
title: **GBB: A Generic Blackboard Development System**
source: AAAI-86
categories: Distributed AI, Knowledge Representation
system: GBB *language:* 7 *Pages*
institution: University of Massachusetts, Amherst *funded by:* NSF

CORK87

author: Corkill,D. Gallagher,K. Johnson,P.

title: **Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures**

source: AAAI-87

categories: Distributed AI, Knowledge Representation

system: GBB *language:* 6 *Pages*

institution: University of Massachusetts, Amherst *funded by:* NSF

CORK88

author: Corkill,D. Gallagher,K.

title: **Tuning a Blackboard-based Application: A case study using GBB**

source: AAAI-88

categories: Distributed AI, Knowledge Representation

system: GBB *language:* 6 *Pages*

institution: University of Massachusetts, Amherst *funded by:* NSF

CORN87

author: Cornell,M.

title: **The KATE Shell - An Implementation of Model-based Control, Monitor and Diagnosis**

source: SOAR-87

categories: Fault Diagnosis, Control

system: KATE *language:* FRL 6 *Pages*

institution: NASA-KSC *funded by:* NASA-KSC

COTT88

author: Cottman,B. Paslay,R.

title: **ACCESS: A Communicating and Cooperating Systems System**

source: DTIC

categories: Distributed AI

system: ACCESS *language:* LISP & LOOPS 11 1 *Pages*

institution: Symbiotics *funded by:* US Army Signals Warfare Center

CRUI87

author: Cruise,A. Ennis,R. Finkel,A. Hellerstein,J. Klein,D. Loeb,D. Masullo,M. Milliken,M. Van Woerken,H. Waite,N.

title: **YES/L1: Integtating Rule-Based, Procedural and Real-time Programming for Industrial Applications**

source: CAIA-87

categories: Real Time Systems

system: YES/L1 *language:* 6 *Pages*

institution: IBM & Univ of Pennsylvania *funded by:* IBM

CRUS87

author: Cruse,B.

title: **TALOS: A Distributed Architecture for Intelligent Monitoring and Anomaly Diagnosis of the Hubble Space Telescope**

source: CAISA-87

categories: Distributed AI, Fault Diagnosis, Real Time Systems

system: TALOS *language:* LES 6 *Pages*

institution: Lockheed *funded by:* NASA-Goddard

DAMB87

author: D'Ambrosio,B. Fehling,M. Forrest,S. Raulefs,P. Wilber,B.

title: **Real-time Process Management for Materials Composition in Chemical Manufacturing**

source: IEEE Expert Vol 2 Num 2 (Summer '87)

categories: Real Time Systems

system: MCM *language:* HCVM 14 *Pages*

institution: FMC & Teknowledge *funded by:*

DASG86

author: Dasgupta,P.

title: **A Probe-Based Monitoring Scheme for an Object-Oriented, Distributed Operating System**

source: OOPSLA-86

categories: Distributed Systems

system: CLOUDS *language:* 10 *Pages*

institution: Georgia Tech *funded by:* NASA

DAVI82

author: Davis,R. Shrobe,H. Hamscher,W. Wieckert,K. Shirley,M. Polit,S.

title: **Diagnosis Based on Description of Structure and Function**

source: AAAI-82

categories: Fault Diagnosis

system: *language:* 6 *Pages*

institution: MIT & DEC *funded by:* DEC

DAVI83

author: Davis,R.

title: **Diagnosis via Causal Reasoning: Paths of Interaction and the Locality Principle**

source: AAAI-83

categories: Fault Diagnosis

system: *language:* 7 *Pages*

institution: MIT *funded by:* DEC

DAY88

author: Day,W.

title: **Cooperative Knowledge Bases**

source: DTIC

categories: Distributed AI
system: *language:* 22 *Pages*
institution: Southeastern Center for Electrical Engineering Education *funded by:* RADC

DURF85

author: Durfee,E. Lesser,V. Corkill,D.
title: **Coherent Cooperation Among Communicating Problem Solvers**
source: ??
categories: Distributed AI
system: DVMT *language:* 46 *Pages*
institution: University of Massachusetts,Amherst *funded by:* NSF

ENSO86

author: Ensor,J. Gabbe,J.
title: **Transactional Blackboards**
source: International Journal for AI in Engineering, Vol 1, Num 2
categories: Distributed AI
system: *language:* LISP & FLAVORS 9 *Pages*
institution: AT&T Bell Labs *funded by:*

ERMA88

author: Erman, Lee D., Jay S. Lark, and Frederick Hayes-Roth
title: **ABE: An Environment for Engineering Intelligent Systems**
source: IEEE Transactions on Software Engineering 14(12), December 1988
categories: Distributed AI
system: ABE *language:* multiple 12 *Pages*
institution: Teknowledge *funded by:* AFSC, DARPA

ESTE86

author: Esteva,J. Reynolds,R.
title: **A Real-time Knowledge Based Expert System For Diagnostic Problem Solving**
source: SPIE Vol 635 (Applications III - 1986)
categories: Real Time Systems, Fault Diagnosis
system: IDD *language:* 9 *Pages*
institution: Wayne State University *funded by:*

FERG87

author: Ferguson,J. Wagner,R.
title: **Beyond Rules: The Next Generation of Expert Systems**
source: SOAR-87
categories: Knowledge Representation
system: PARAGON *language:* 7 *Pages*
institution: Ford Aerospace *funded by:*

FIKE85

author: Fikes,R. Kehler,T.
title: **The Rule of Frame-Based Representation in Reasoning**
source: CACM Volume 28, No. 9 (1985)
categories: Knowledge Representation
system: *language:* KEE 17 *Pages*
institution: Intellicorp *funded by:*

FOX83

author: Fox,M. Lowenfield,S. Kleinosky,P.
title: **Techniques for Sensor-Based Diagnosis**
source: IJCAI-83
categories: Fault Diagnosis
system: PDS *language:* SRL 5 *Pages*
institution: CMU & Westinghouse *funded by:* Westinghouse

FOX88

author: Fox,M. Husain,N. McRoberts,M. Reddy,Y.
title: **Knowledge Based Simulation: An Artificial Intelligence Approach to System Modelling and Automating the Simulation Life Cycle**
source: Report RI-TR-88-5
categories: Simulation, Fault Diagnosis
system: KBS *language:* CRL 41 *Pages*
institution: CMU *funded by:* DEC

FRIE85

author: Friedman,L.
title: **Controlling Production Firing: The FCL Language**
source: IJCAI-85
categories: Fault Diagnosis
system: FAITH *language:* 8 *Pages*
institution: JPL *funded by:* NASA

FRIE87

author: Friedman,L.
title: **FAITH: A Diagnostic Shell**
source: AI-87 (Long Beach)
categories: Fault Diagnosis
system: FAITH *language:* LISP 10 *Pages*
institution: ISI *funded by:*

GADB88

author: Gadbois,L.
title: **OPS83: Style Guide for High Performance**
source: DTIC
categories:

system: *language:* 25 *Pages*

institution: Naval Ocean Systems Center *funded by:*

GALL85

author: Gallanti,M. Guida,G. Spampinato,L. Stefanini,A.

title: **Representing Procedural Knowledge in Expert Systems: An Application to Process Control**

source: IJCAI-85

categories: Control, Knowledge Representation

system: PROP *language:* LISP 0 *Pages*

institution: CISE, Italy *funded by:* ENEL-DSR

GALL87

author: Galliher,J.

title: **A Comparison of the Rule/Frame Approach used in Picon with the Description of Structure and Function used in Kate**

source: SOUTHCON-87

categories: Fault Diagnosis, Control

system: KATE *language:* LISP 3 *Pages*

institution: NASA-KSC *funded by:* NASA

GASS85

author: Gasser,L. Braganza,C. Herman,N.

title: **MACE: A Flexible Testbed for Distributed AI Research**

source: Huhns

categories: Distributed AI, Knowledge Representation

system: MACE *language:* LISP 33 *Pages*

institution: USC *funded by:* Lawrence Livermore Laboratory

GASS87b

author: Gasser,L.

title: **A Multi-Agent Intelligent Diagnostic System**

source: DAI Research Note 28

categories: Distributed AI, Fault Diagnosis

system: MACE *language:* LISP 11 *Pages*

institution: USC *funded by:* Lawrence Livermore Laboratory

GASS88

author: Gasser,L. Braganza,C. Herman,N.

title: **Implementing Distributed AI Systems Using MACE**

source: Bond-Gasser

categories: Distributed AI

system: MACE *language:* LISP 6 *Pages*

institution: USC *funded by:* Lawrence Livermore Laboratory

GENE82

author: Genesereth,M.
title: **Diagnosis Using Heirarchical Design Models**
source: AAAI-82
categories: Fault Diagnosis
system: DART *language:* LISP 6 *Pages*
institution: Stanford *funded by:*

GINS85

author: Ginsberg,M.
title: **Counterfactuals**
source: IJCAI-85
categories: Knowledge Representation, Fault Diagnosis
system: *language:* 5 *Pages*
institution: Stanford *funded by:* ONR

GINS86

author: Ginsberg,M.
title: **Multi-valued logics**
source: AAAI-86
categories: Knowledge Representation
system: *language:* 5 *Pages*
institution: Stanford *funded by:* ONR

GOOD83

author: Goodson,J. Zachary,W. Deimler,J. Stokes,J. Weiland,W. Hopson,J.
title: **Distributed Intelligence Systems: AI Approaches To Cooperative Man-Machine Problem Solving in C3I**
source: AIAA
categories: Distributed AI
system: *language:* NINA 8 *Pages*
institution: Analytics Inc. & Naval Air Development Center *funded by:* NADC

GRAH87

author: Graham,G.
title: **Real-World Distributed Databases**
source: Unix Review, May 1987
categories: Database Systems, Distributed Systems
system: *language:* 7 *Pages*
institution: Gartner Group *funded by:*

GRAY87

author: Gray,J.
title: **Transparency in its Place**
source: Unix Review, May 1987
categories: Database Systems, Distributed Systems
system: *language:* 9 *Pages*

institution: Tandem *funded by:*

GREE85

author: Green,P.

title: **AF: A Framework for Real-Time Distributed Cooperative Problem Solving**

source: Huhns

categories: Distributed AI, Real Time Systems

system: AF *language:* C & LISP 23 *Pages*

institution: Worcester Polytechnic Institute *funded by:*

GREE86

author: Green,P.

title: **Resource Limitation Issues in Real-time Intelligent Systems**

source: SPIE Vol 635 (Applications III - 1986)

categories: Real Time Systems, Distributed AI

system: DSN *language:* AF (Precursor) 8 *Pages*

institution: Worcester Polytechnic Institute *funded by:*

GREE87

author: Green,P. Glasson,D. Pomarede,J. Acharya,N.

title: **Real-Time Artificial Intelligence Issues in the Development of the Adaptive Tactical Navigator**

source: SOAR-87

categories: Real Time Systems, Distributed AI

system: ATN *language:* AF 8 *Pages*

institution: Worcester Poly & Analytic Sciences *funded by:* AFAL

GRIE84

author: Greismer,J. Hong,S. Karnaugh,M. Kastner,J. Schor,M. Ennis,R. Klein,D. Milliken,M. Van Woerken,H.

title: **YES/L1: Integtating Rule-Based, Procedural and Real-time Programming for Industrial Applications**

source: AAAI-84

categories: Real Time Systems, Control

system: YES/MVS *language:* OPS5 7 *Pages*

institution: IBM *funded by:* IBM

GUPT83

author: Gupta,A. Forgy,C.

title: **Measurements on Production Systems**

source: Report CS-83-167

categories:

system: *language:* 30 *Pages*

institution: CMU *funded by:* DARPA (AFAL)

GUPT84

author: Gupta,N. Seviora,R.

title: **An Expert System Approach to Real Time Sytem Debugging**

source: CAIA-84

categories: Real Time Systems

system: *language:* Prolog 8 *Pages*

institution: University of Waterloo *funded by:* NSRC of Canada

HALS78

author: Halstead, Robert H.

title: **Object Management on Distributed Systems**

source: Proceedings of the 7th Texas Conference on Computing Systems, 1978

categories: Distributed Systems, Object Oriented

system: MuNet *language:* 8 *Pages*

institution: MIT *funded by:*

HANK85

author: Hankins,G. Jordan,J. Katz,J. Mulvehill,A. Dumoulin,J. Ragusa,J.

title: **Expert Mission Planning and Replanning Scheduling System**

source: ESIG-85

categories: Planning

system: EMPRESS *language:* FLAVORS 7 *Pages*

institution: MITRE & NASA-KSC *funded by:* NASA

HART85

author: Hartzband,D.

title: **The Anatomy of Knowledge Base Management**

source: AI-85 (Long Beach)

categories: Database Systems

system: *language:* 8 *Pages*

institution: DEC *funded by:*

HAWK85

author: Hawkinson,L. Levin,M. Knickerbocker,C. Moore,R.

title: **A Paradigm for Real-Time Inference**

source: AI-85 (Long Beach)

categories: Real Time Systems

system: PICON *language:* LISP 6 *Pages*

institution: LMI *funded by:*

HAYE84

author: Hayes-Roth,B.

title: **BB1: An architecture for blackboard systems that control, explain, and learn about their own behavior**

source: Report No. HPP 84-16

categories: Distributed AI, Knowledge Representation

system: BB1 *language:* 22 *Pages*

institution: Stanford *funded by:*

HAYE87

author: Hayes-Roth,B.

title: **A Multi-Processor Interrupt-Driven Architecture for Adaptive Intelligent Systems**

source: Report No. KSL 87-31

categories: Distributed AI, Real Time Systems, Knowledge Representation

system: MI *language:* 19 *Pages*

institution: Stanford *funded by:* Rockwell International

HEIL87

author: Heiler,S.

title: **Support for Heterogeneous Databases**

source: Unix Review, May 1987

categories: Database Systems, Distributed Systems

system: *language:* 10 *Pages*

institution: Computer Corp of America *funded by:*

HEWI73

author: Hewitt,C. Bishop,P. Steiger,R.

title: **A Universal Modular ACTOR Formalism for Artificial Intelligence**

source: IJCAI-73

categories: Distributed AI, Distributed Systems, Knowledge Representation

system: ACTORS *language:* 11 *Pages*

institution: MIT *funded by:* ONR

HEWI83

author: Hewitt,C. deJong,P.

title: **Analyzing the Roles of Descriptions and Actions in Open Systems**

source: AAAI-83

categories: Distributed AI, Distributed Systems

system: ACTORS *language:* ACT2 6 *Pages*

institution: MIT *funded by:* System Development Foundation

HEWI85

author: Hewitt,C.

title: **The Challenge of Open Systems**

source: BYTE, April 1985

categories: Distributed AI

system: ACTORS *language:* 11 *Pages*

institution: MIT *funded by:* System Development Foundation

HEWI86

author: Hewitt,C.

title: **Concurrency in Intelligent Systems**

source: AI EXPERT, Premier, 1986
categories: Distributed AI, Distributed Systems
system: ACTORS *language:* 5 *Pages*
institution: MIT *funded by:*

HILL86

author: Hill,R.
title: **Prospective Reasoning ABout Liaisons Between Agents**
source: DAI Group Research Note 22
categories: Distributed AI
system: *language:* 21 *Pages*
institution: USC *funded by:*

HO86

author: Ho,E.
title: **Simulating a Connectionist Model of Scene Analysis in a Multi-Agent Environment**
source: DAI Research Note ?
categories: Distributed AI, Vision, Parallelism
system: MACE *language:* LISP 13 *Pages*
institution: USC *funded by:*

HOOD86

author: Hood, S. Mason, K.
title: **Knowledge-based Systems for Real-time Applications**
source: Applications of Expert Systems, ed. Quinlan, 1988
categories: Real Time SystemsSimulastion, Simulation, Classification
system: ESM *language:* Prolog & C 18 *Pages*
institution: Australian DOD *funded by:* Australian DOD

ISHI87

author: Ishikawa, Yutaka and Mario Tokoro
title: **Orient84/K: AN Object Oriented Concurrent Programming Language for Knowledge Represetnation**
source: Object Oriented Concurrent Programming, Yonezawa & Tokoro, eds, MIT Press, 1987
categories: Distributed AI, Knowledge Representation, Object Oriented
system: Orient84/K *language:* Orient84/K 40 *Pages*
institution: *funded by:*

JOHA87

author: Johannes,J. Carnes,J.
title: **Qualitative Models for Planning: A Gentle Introduction**
source: CAISA-87
categories: Planning, Temporal Reasoning
system: *language:* 6 *Pages*
institution: Univ of Alabama *funded by:*

JOHN87

author: Johnson, Herrick J.
title: **"Each Piece in its Place"**
source: Unix Review, June 1987
categories: Distributed Systems
system: NCS *language:* C 9 *Pages*
institution: Apollo *funded by:* Apollo

JONE86

author: Jones, Michael B. and Richard F. Rashid
title: **Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems**
source: OOPSLA '86 Proceedings
categories: Distributed Systems, Object Oriented
system: Mach *language:* multiple 11 *Pages*
institution: Carnegie-Mellon *funded by:* DARPA

KACP85

author: Kacprzyk,J. Yager,R.
title: **Emergency-Oriented Expert Systems: A Fuzzy Approach**
source: Information Sciences Vol 37, 1985
categories: Real Time Systems, Control
system: *language:* 13 *Pages*
institution: Hagan School of Business, Iona College *funded by:*

KAEM87

author: Kaemmerer,W. Allard,J.
title: **An Automated Reasoning Technique for Providing Moment-by-Moment Advice Concerning The Operation of a Process**
source: AAAI-87
categories: Fault Diagnosis, Real Time Systems
system: PSMS *language:* LISP 5 *Pages*
institution: Honeywell *funded by:*

KAIS87

author: Kaiser, Gail E. and David Garlan
title: **MELDing Data Flow and Object Oriented Programming**
source: OPSLA '87 Proceedings, October 1987
categories: Object Oriented
system: MELD *language:* MELD 14 *Pages*
institution: Columbia University, Carnegie Mellon *funded by:* AT&T, U.S. Army

KARS88

author: Karsai,G. Biegl,C. Sztipanovits,J. Kawamura,K. Miyasaka,N. Inui,M.
title: **Knowledge-based Approach to Real-time Supervisory Control**
source: American Control Conference, 1988

categories: Real Time Systems, Distributed AI, Control
system: IPCS *language:* Multigraph 6 *Pages*
institution: Vanderbilt U. & Osaka Gas Co. *funded by:*

KHOS86

author: Khoshafian, S.N. & G. P. Copeland
title: **Object Identity**
source: OOPSLA '86 Proceedings
categories: Object Oriented Systems
system: *language:* 11 *Pages*
institution: MCC, Austin TX *funded by:*

KING82

author: King,J.
title: **Artificial Intelligence Techniques for Device Troubleshooting**
source: Technical Note CSL-82-9
categories: Fault Diagnosis
system: *language:* 19 *Pages*
institution: Hewlett Packard *funded by:*

LAFF86

author: Laffey,T. Perkins,W. Nguyen,T.
title: **Reasoning About Fault daignosis with LES**
source: IEEE Expert, Spring 86
categories: Fault Diagnosis
system: LES *language:* 8 *Pages*
institution: Lockheed *funded by:*

LAFF88a

author: Laffey,T. Cox,P. Schmidt,J. Kao,S. Read,J.
title: **Real-Time Knowledge-Based Systems**
source: AI Magazine, Spring 1988
categories: Real Time Systems
system: *language:* 19 *Pages*
institution: Lockheed *funded by:*

LAFF88b

author: Laffey,T. Weitzenkamp,J. Read,J. Kao,S. Schmidt,J.
title: **Intelligent Real-Time Monitoring**
source: AAAI-88
categories: Real Time Systems, Distributed AI, Fault Diagnosis
system: L*STAR *language:* C 5 *Pages*
institution: Lockheed *funded by:* NASA-Goddard

LASK87

author: Laskowski,S. Hofmann,E.

title: **Script-Based Reasoning For Situation Monitoring**

source: AAAI-87

categories: Control, Temporal Reasoning, Knowledge Representation

system: SCAN *language:* 5 *Pages*

institution: MITRE *funded by:* RADC

LE88

author: Le,T. Homeier,P.

title: **Portable Inference Engine: An Extended CLIPS for Real-Time Productions**

source: SOAR-88

categories: Real Time Systems

system: PIE *language:* CLIPS 6 *Pages*

institution: Aerospace *funded by:*

LEAO88

author: Leao, L. Talukdar, S.

title: **COPS: A System for Constructing Multiple Blackboards**

source: Bond & Gasser

categories: Distributed AI

system: COPS *language:* OPS5 9 *Pages*

institution: CMU *funded by:* CMU & Banco Itau S.S., Brazil

LEIN86a

author: Leinweber,D.

title: **Controlling real-time processes on the space station with expert systems**

source: SPIE Vol 729 (Space Station II - 1986)

categories: Real Time Systems, Control

system: PICON *language:* LISP 19 *Pages*

institution: LMI *funded by:*

LEIN86b

author: Leinweber,D. Gidwani,K.

title: **Real-Time Expert System Development Techniques and Applications**

source: WESTEX-86

categories: Real Time Systems, Control

system: PICON *language:* LISP 9 *Pages*

institution: LMI *funded by:*

LEIN87

author: Leinweber,D.

title: **Expert Systems in Space**

source: IEEE Expert, Spring 87

categories: Real Time Systems, Control

system: PICON *language:* LISP 11 *Pages*
institution: LMI *funded by:*

LESS77a

author: Lesser,V. Hayes-Roth,F.
title: **Focus of Attention in the Hearsay-II Speech Understanding System**
source: IJCAI-77
categories: Distributed AI, Knowledge Representation
system: HEARSAY-II *language:* 9 *Pages*
institution: University of Massachusetts & RAND *funded by:* CMU, DARPA (Air Force Office of Scientific Research)

LESS77b

author: Lesser,V. Erman,L.
title: **A Retrospective View of the Hearsay-II Architecture**
source: IJCAI-77
categories: Distributed AI, Knowledge Representation
system: HEARSAY-II *language:* 12 *Pages*
institution: University of Massachusetts & CMU *funded by:* DARPA (Air Force Office of Scientific Research)

LESS80

author: Lesser,V. Erman,L.
title: **Distributed Interpretation: A Model and Experiment**
source: IEEE Transcations on Computers, Vol 29, No. 12 (Dec. 1980)
categories: Distributed AI
system: HEARSAY-II *language:* 20 *Pages*
institution: University of Massachusetts,Amherst *funded by:* DARPA

LESS81

author: Lesser,V. Corkill,D.
title: **Functionally Accurate, Cooperative Distributed Systems**
source: IEEE Transcations on SMC, Vol 11, No 1 (Jan 81)
categories: Distributed AI
system: DVMT *language:* 16 *Pages*
institution: University of Massachusetts,Amherst *funded by:* NSF

LESS83

author: Lesser,V. Corkill,D.
title: **The Distributed Vehicle Monitoring Testbed**
source: AI Magazine, Fall 1983
categories: Distributed AI
system: DVMT *language:* 19 *Pages*
institution: University of Massachusetts,Amherst *funded by:* NSF

LESS88

author: Lesser,V. Pavlin,J. Durfee,E.
title: **Approximate Processing in Real-Time Problem Solving**
source: AI Magazine, Spring 88
categories: Distributed AI, Real Time Systems, Temporal Reasoning
system: DVMT *language:* 13 *Pages*
institution: University of Massachusetts,Amherst *funded by:* ONR

LEVI87

author: Levi,S. Agrawala,A.
title: **Temporal Relations and Structures in Real-Time Operating Systems**
source: Technical Report TR-87-61
categories: Real Time Systems, Temporal Reasoning
system: *language:* 48 *Pages*
institution: University of Maryland *funded by:* ONR

LEWI88

author: Lewis, S.
title: **Closely Watched Brains: Expert Systems Running in Real Time**
source: AI-88 (Long Beach)
categories: Real Time Systems, Control
system: *language:* 6 *Pages*
institution: USC *funded by:* Aerospace Corp.

LIEB88

author: Lieberherr, K., I. Holland, and A. Riel
title: **Object-Oriented Programming: An Objective Sense of Style**
source: OOPSLA '88
categories: Object Oriented
system: Demeter *language:* LISP/C++ 12 *Pages*
institution: Northeastern University *funded by:*

LIND87

author: Linton, Mark A.
title: **Distributed Management of a Software Database**
source: IEEE Software, November 1987
categories: Distributed Systems, Object Oriented
system: Allegro *language:* C++ 7 *Pages*
institution: Stanford *funded by:* DEC

LISK83

author: Liskov, Barbara and Robert Scheifler
title: **Guardians and Actions: Linguistic Support for Robust, Distributed Programs**
source: ACM Transactions on Programming Languages and Systems 6(3), July 1983
categories: Distributed Systems
system: ARGUS *language:* ARGUS 24 *Pages*

institution: MIT *funded by:* DARPA

LIZZ88

author: Lizza,C. Friedlander,C.

title: **The Pilot's Associate: A Forum for the Integration of Knowledge Based Systems and Avionics**

source: NAECON-88

categories: Distributed AI

system: *language:* ART 7 *Pages*

institution: AFWAL & BDM *funded by:* DARPA

MALE85

author: Maletz,M.

title: **Navex: Space Shuttle Navigation Expert System**

source: AI-85 (Long Beach)

categories: Real Time Systems, Fault Diagnosis

system: NAVEX *language:* ART 6 *Pages*

institution: NASA-JSC *funded by:* NASA

MALE87

author: Maletz,M.

title: **An Architecture for Consideration of Multiple Faults**

source: CAIA-85

categories: Fault Diagnosis

system: *language:* ART 8 *Pages*

institution: NASA-JSC *funded by:* NASA

MARS87

author: Marshall,P.

title: **Expert System Applications in Spacecraft Subsystem Controllers**

source: SOAR-87

categories: Control

system: *language:* 5 *Pages*

institution: NASA-JSC *funded by:* NASA

MCAR82

author: McArthur,D. Steeb,R. Cammarata,S.

title: **A Framework For Distributed Problem Solving**

source: AAAI-82

categories: Distributed AI

system: DATC *language:* LISP 4 *Pages*

institution: Rand *funded by:*

MCCO87

author: McCord,R. Hanner,M.

title: **Connecting Islands of Information**

source: Unix Review, May 1987
categories: Database Systems, Distributed Systems
system: INGRES language: 7 Pages
institution: Relational Technology funded by:

MCCU87

author: McCullough, Paul L.
title: **Transparent Forwarding: First Steps**
source: OOPSLA '87
categories: Object Oriented, Distributed Systems
system: language: 11 Pages
institution: Xerox PARC funded by: Xerox

MEYE88

author: Meyer, Bertrand
title: **Object Oriented Software Construction**
source: Object Oriented Software Construction, Prentice Hall, 1988
categories: Object Oriented Systems
system: Eiffel language: Eiffel 53 4 Pages
institution: Interactive Software Engineering funded by:

MILL88

author: Miller,R.
title: **Inside Expert Process Control**
source: Managing Automation, January 88
categories: Real Time Systems, Control
system: G2 & PICON language: LISP 5 Pages
institution: GIGAMOS & GENSYM funded by:

MOOR84

author: Moore,R. Hawkinson,L. Knickerbocker,C. Churchman,L.
title: **A Real-Time Expert System for Process Control**
source: CAIA-84
categories: Real Time Systems, Control, Fault Diagnosis
system: PICON language: LISP 8 Pages
institution: LMI funded by:

MRAZ87

author: Mraz,R.
title: **Performance Analysis of Parallel Branch and Bound Search with the Hypercube Architecure**
source: SOAR-87
categories: Parallelism
system: language: C 8 Pages
institution: USAF-JSC funded by: USAF

MULL87

author: Mullikin,R.

title: **A Framework for Real-Time Distributed Expert Systems: On-Orbit Spacecraft Fault Diagnosis, Monitoring and Control**

source: CAISA-87

categories: Real Time Systems, Distributed AI

system: *language:* 5 *Pages*

institution: ARINC Research Corp *funded by:* JPL (NASA?)

NEW87

author: New,E.

title: **Knowledge-based Control and Redundancy Management Techniques Used in NASA's Kate Project**

source: SOUTHCON-87

categories: Fault Diagnosis, Control

system: KATE *language:* LISP 8 *Pages*

institution: NASA-KSC *funded by:* NASA

NEW88

author: New,E.

title: **A Complete Knowledge-based Approach to Process Control**

source: SOUTHCON-88

categories: Fault Diagnosis, Control

system: KATE *language:* LISP 4 *Pages*

institution: NASA-KSC *funded by:* NASA

NG88

author: Ng,A.

title: **A Cooperative Expert System Architecture for Embedded Avionics**

source: NAECON-88

categories: Distributed AI, Real Time Systems

system: ESP/HMOSE *language:* 6 *Pages*

institution: Honeywell *funded by:*

NGUY86

author: Nguyen,T. Chiou,W.

title: **Cooperating Expert Systems for Space Station Power Distribution Management**

source: SPIE Vol 729 (Space Station II 1986)

categories: Distributed AI

system: CARTS *language:* ART 4 *Pages*

institution: Lockheed *funded by:*

NIER87

author: Nierstrasz, G. M.

title: **Active Objects in Hybrid**

source: OOPSLA '87 Proceedings
categories: Distributed Computing, Object Oriented
system: Hybrid *language:* Hybrid 11 *Pages*
institution: Centre Universitaire d'Informatique, Universite de Geneve *funded by:* National Science & Eng. Rsrch Council of Canada

NII79a

author: Nii,P. Feigenbaum,E.
title: **Rule-based Understanding of Signals**
source: "Pattern Directed Inf. Systems"
categories: Knowledge Representation
system: SU/X & SU/P *language:* 19 *Pages*
institution: Stanford *funded by:* DARPA

NII79b

author: Nii,P. Aiello,N.
title: **AGE (Attempt to Generalize): A Knowledge-Based Program for Knowledge-Based Programs**
source: IJCAI-79
categories: Knowledge Representation
system: AGE *language:* 11 *Pages*
institution: Stanford *funded by:* DARPA

NII82

author: Nii,P. Feigenbaum,E. Anton,J. Rockmore,A.
title: **Signal-to-Symbol Transformationnal: HASP/SIAP Case Study**
source: AI Magazine Spring 82
categories: Knowledge Representation
system: HASP & SIAP *language:* 13 *Pages*
institution: Stanford & Systems Control Technology *funded by:*

NII86

author: Nii,P.
title: **Blackboard Systems**
source: AI Magazine Volume 7, nos. 3 and 4
categories: Knowledge Representation
system: *language:* 42 *Pages*
institution: Stanford *funded by:* DARPA

NUGE88

author: Nugent,R. Tucker,R.
title: **TAC-1: A Knowledge-based Air Force Tactical Battle Management Testbed**
source: DTIC
categories: Distributed AI
system: TAC-1 *language:* LISP & FLAVORS 60 *Pages*

institution: MITRE *funded by:* RADC

OREI85

author: O'Reilly,C. Cromarty,A.
title: **"Fast" is not "real-time": Designing effective real-time AI systems**
source: SPIE Vol 548 (Applications II - 1985)
categories: Real Time Systems
system: *language:* 9 *Pages*
institution: Advanced Information & Decision Systems *funded by:*

PATE85

author: Paterson,A. Sachs,P. Turner,M.
title: **ESCORT: The Application of Causal Knowledge To Real -Time Process Control**
source: BCSES-85
categories: Real Time Systems, Control
system: ESCORT *language:* 9 *Pages*
institution: PA Computers & Telecommunications, England *funded by:*

PEAR85

author: Pearson,G.
title: **Mission Planning Within the Framework of the Blackboard Model**
source: ESIG-85
categories: Planning
system: *language:* BB1/BEDS 6 *Pages*
institution: FMC *funded by:*

PIER87

author: Pierce,R.
title: **Expert Mission Planning and Replanning Scheduling System for NASA KSC Payload Operations**
source: SOAR-87
categories: Planning
system: EMPRESS *language:* LISP & FLAVORS 8 *Pages*
institution: NASA-KSC *funded by:* NASA

RAMA84

author: Ramamritham,K. Stankovic,J.
title: **Dynamic Task Scheduling in Hard Real-Time Distributed Systems**
source: IEEE Software, July 1984
categories: Real Time Systems
system: *language:* 11 *Pages*
institution: University of Massachusetts,Amherst *funded by:* US Army & NSF

RAUL87

author: Raulefs,P. D'Ambrosio,B. Fehling,M. Forrest,S. Wilber,B.
title: **Real-time Process Management for Materials Composition**
source: CAIA-87
categories: Real Time Systems
system: MCM *language:* HCVM 6 *Pages*
institution: FMC & Teknowledge *funded by:*

READ86

author: Read,J. Howland,T. Perkins,W.
title: **Use of Communicating Expert Systems in Fault Diagnosis for Space Station Applications**
source: SPIE Vol 729 (Space Station II - 1986)
categories: Distributed AI, Fault Diagnosis, Knowledge Representation
system: *language:* LES 10 *Pages*
institution: Lockheed *funded by:*

READ88

author: Read,J. Schmidt,J. Kao,S. Laffey,T.
title: **Real-Time Knowledge-Based Monitoring of Telemetry Data**
source: SPIE Vol 937 (Applications VI - 1988)
categories: Real Time Systems, Fault Diagnosis
system: L*STAR *language:* 8 *Pages*
institution: Lockheed *funded by:* NASA-Goddard

REGG83

author: Reggia,J. Nau,D. Wang,P.
title: **A New Inference Method for Frame-Based Expert Systems**
source: AAAI-83
categories: Fault Diagnosis
system: GSC *language:* 5 *Pages*
institution: University of Maryland *funded by:* NINCDS

REYN88

author: Reynolds,D.
title: **MUSE: A Toolkit for Embedded, Real-time AI**
source: Engelmore-Morgan
categories: Distributed AI, Real Time Systems
system: MUSE *language:* PopTalk 13 *Pages*
institution: *funded by:* Royal Aircraft Establishment (England)

RILE87

author: Riley,G.
title: **Implementing CLIPS on a Parallel Computer**
source: SOAR-87
categories: Parallelism

system: *language:* CLIPS 5 *Pages*
institution: NASA-JSC *funded by:* NASA

ROWL87

author: Rowley, S. Shrobe, H. Cassels, R. Hamscher, W.
title: **Joshua: Uniform Access to Heterogeneous Knowledge Structures**
source: AAAI-87
categories: Knowledge Representation
system: Joshua *language:* LISP & Flavors 5 *Pages*
institution: Symbolics & MIT *funded by:*

SACR88

author: Scarl,E. Jamieson,J. New,E.
title: **Deriving Fault Location and Control from a Functional Model**
source: 3rd IEEE Symposium on Intelligent Control, 1988
categories: Fault Diagnosis, Control
system: KATE *language:* 16 *Pages*
institution: Boeing & NASA-KSC *funded by:* NASA & Boeing

SAGE87

author: Sage,A.
title: **Information Systems Engineering for Distributed Decisionmaking**
source: IEEE SMC Vol 17, No 6, November/December 1987
categories: Distributed AI, Distributed Systems, Real Time Systems
system: *language:* 17 *Pages*
institution: George Mason University *funded by:*

SCAR85

author: Scarl,E. Jamieson,J. Delaune,C.
title: **A Fault Detection and Isolation Method Applied To Liquid Oxygen Loading For the Space Shuttle**
source: IJCAI-85
categories: Fault Diagnosis
system: LES *language:* FRL 3 *Pages*
institution: Mitre & NASA-KSC *funded by:* NASA-KSC

SCAR87

author: Scarl,E. Jamieson,J. Delaune,C.
title: **Diagnosis and Sensor Validation through Knowledge of Structure and Fuction**
source: IEEE Transations on SMC, Vol 17 No 3, May/June 1987
categories: Fault Diagnosis
system: LES *language:* 9 *Pages*
institution: Boeing & NASA-KSC *funded by:* NASA & Boeing

SCHO84

author: Schor,M.

title: **Using Declarative Knowledge Representation Techniques: Implementing Truth Maintenance in OPS5**

source: CAIA-84

categories: Real Time Systems, Knowledge Representation

system: YES/MVS *language:* OPS5 6 *Pages*

institution: IBM *funded by:* IBM

SCHU87a

author: Schudy,R.

title: **A Conceptual Framework for Intelligent Real Time Information Processing**

source: SOAR-87

categories: Real Time Systems, Knowledge Representation

system: *language:* 5 *Pages*

institution: BBN *funded by:*

SCHU87b

author: Schultz, R. & A. Cardenas

title: **An Approach and Mechanism for Auidtable and Testable Advanced Transaction Processing Systems**

source: IEEE Transactions on Software Engineering, SE-13 (6), June 1987

categories: Real Time Systems, Distributed Systems, Expert Systems

system: *language:* 11 *Pages*

institution: UCLA *funded by:*

SCHU87c

author: Schultz, R. & A. Cardenas

title: **An Expert System Shell for Dynamic Auditing in a Distributed Environment**

source: ACM SIGSAC '87 Conference Proceedings

categories: Real Time Systems, Distributed Systems, Expert Systems

system: *language:* 11 *Pages*

institution: UCLA *funded by:*

SCHW87

author: Schwan, Karsten, Prabha Gopinath, and Win Bo

title: **CHAOS – Kernel Support for Objects in the Real-Time Domain**

source: IEEE Transactions on Computers C-36(8), August 1987

categories: Real Time Systems, Object Oriented

system: GEM *language:* 13 *Pages*

institution: Ohio State University *funded by:* DARPA

SEWA87

author: Seward,W.

title: **Task Allocation in a Distributed Computing System**

source: SOAR-87

categories: Distributed Systems
system: *language:* 9 *Pages*
institution: Air Force Institute of Technology, Wright-Patterson *funded by:*

SHAR88

author: Sharma,D. Sridharan,N.
title: **Knowledge-Based Real-Time Control: A Parallel Processing Perspective**
source: AAAI-88
categories: Real Time Systems, Parallelism, Control
system: QP-Net *language:* HCVM 6 *Pages*
institution: FMC *funded by:*

SHEE87

author: Sheets, Kitrick B. and Kwei-Jay Lin
title: **A Kernel Level Remote Procedure Call Mechanism**
source:
categories: Distributed Systems
system: Xinu *language:* 7 *Pages*
institution: Univ. of Illiniois *funded by:*

SHRI87

author: Shrivastava,S. Dixon,G. Parrington,G.
title: **Objects and Actions in Reliable Distributed Systems**
source: NTIS
categories: Distributed Systems
system: *language:* 21 *Pages*
institution: Newcastle upon Tyne Univ. (England) *funded by:* SERC/Alvey

SHRO88

author: Shrobe, H. Aspinall, J. Mayle, N..
title: **Towards a Virtual Parallel Inference Engine**
source: AAAI-88
categories: Knowledge Representation, Parallelism
system: Joshua *language:* LISP & Flavors 6 *Pages*
institution: Symbolics & MIT *funded by:*

SIEM86

author: Siemens,R. Golden,M. Ferguson,J.
title: **StarPlan II: Evolution of an Expert System**
source: AAAI-86
categories: Distributed AI, Real Time Systems, Fault Diagnosis
system: STARPLAN *language:* KEE 7 *Pages*
institution: Ford Aerospace *funded by:*

SKAP86

author: Skapura,D. Zoch,D.
title: **A Real-time Production System for Telemetry Analysis**
source: ESGS-86
categories: Real Time Systems
system: HEAT *language:* OPS5 7 *Pages*
institution: Ford Aerospace *funded by:* NASA-JSC

SLOM85

author: Sloman,A.
title: **Real Time Multiple-Motive Expert Systems**
source: BCSES-85
categories: Real Time Systems, Parallelism
system: *language:* 11 *Pages*
institution: University of Sussex, England *funded by:* GEC & Renaissance Trust

SMIT81

author: Smith,R. Davis,R.
title: **Frameworks for Cooperation in Distributed Problem Solving**
source: IEEE Transactions on SMC, Vol 11 No 1 (Jan 1981)
categories: Distributed AI
system: *language:* 10 *Pages*
institution: DREA & MIT *funded by:* Nat. Defence, Canada & DARPA (ONR)

SMIT85

author: Smith,S. Ow,P.
title: **The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks**
source: IJCIA-85
categories: Planning
system: ISIS *language:* 3 *Pages*
institution: CMU *funded by:* Air Force & Westinghouse

SPEC82

author: Spector, Alfred Z.
title: **Performing Remote Operations Efficiently on a Local Computer Network**
source: CACM 25(4), April 1982
categories: Distributed Systems
system: *language:* 15 *Pages*
institution: Stanford University *funded by:* Xerox

STAL77

author: Stallman,R. Sussman,G.
title: **Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis**

source: Artificial Intelligence 9 (1977)
categories: Fault Diagnosis
system: EL *language:* ARS 62 *Pages*
institution: MIT *funded by:* DARPA (ONR)

STAN88

author: Stankovic,J.
title: **Misconceptions About Real-Time Computing**
source: IEEE Computer, October 1988
categories: Real Time Systems
system: *language:* 10 *Pages*
institution: University of Massachusetts,Amherst *funded by:* ONR

STEE84

author: Steeb,R. McArthur,D. Cammarata,S. Narain,S. Giarla,W.
title: **Distributed Problem Solving for Air Fleet Control: Framework and Implementation**
source: Tech Report N-2139-ARPA
categories: Distributed AI
system: DATC *language:* LISP 42 *Pages*
institution: Rand *funded by:* ARPA

SUSS80

author: Sussman,G. Steele,G.
title: **CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions**
source: Artificial Intelligence 14 (1980)
categories: Fault Diagnosis, Knowledge Representation
system: *language:* 39 *Pages*
institution: MIT *funded by:* NSF

SZTI86

author: Sztipanovits,J. Karsai,G. Biegl,C. Padalkar,S. Purves,R. Williams,R. Christiansen,T.
title: **Programming model for coupled intelligent systems in distributed execution environments**
source: SPIE Vol 729 (Space Station II,1986)
categories: Real Time Systems, Distributed AI, Control
system: Multigraph *language:* LISP 9 *Pages*
institution: Vanderbilt U. & Boeing *funded by:*

SZTI88

author: Sztipanovits,J. Karsai,G. Biegl,C.
title: **Graph Model-Based Approach to the Representation, Interpretation and Execution of Signal Processing Systems**
source: International Journal of Intelligent Systems, Vol3, 269-280
categories: Real Time Systems, Distributed AI, Control

system: Multigraph *language:* LISP 12 *Pages*
institution: Vanderbilt U. *funded by:* Boeing & IBM

TANE85

author: Tanenbaum, A. Van Renesse,R.
title: **Distributed Operating Systems**
source: Computing Surveys, Dec. 1985
categories: Distributed Systems
system: *language:* 52 *Pages*
institution: Vrje U. Amsterdam *funded by:*

TATE85

author: Tate,A.
title: **A Review of Knowledge-Based Planning Techniques**
source: BCSES-85
categories: Planning
system: *language:* 20 *Pages*
institution: University of Edinburgh, Scotland *funded by:*

TENN81

author: Tenney, R. Sandell, N.
title: **Strategies for Distributed Decisionmaking**
source: Bond & Gasser
categories: Distributed AI
system: *language:* 12 *Pages*
institution: MIT & Alphatech *funded by:* NSF & ONR

TOTH87

author: Toth-Fejel,T. Heher,D.
title: **Temporal and Contextual Knowledge in Model-Based Expert Systems**
source: CAISA-87
categories: Knowledge Representation, Temporal Reasoning
system: PARAGON *language:* LISP 5 *Pages*
institution: Ford Aerospace *funded by:*

VERE83

author: Vere,S.
title: **Planning in Time: Windows and Durations for Activities and Goals**
source: IEEE PAMI Vol 5 No 3, May 1983
categories: Planning
system: DEVISER *language:* 22 *Pages*
institution: JPL *funded by:* NASA

WEIS86

author: Weisbin,C. de Saussure,G. Kammer,D.

title: **A real-time expert system for control of an autonomous robot including diagnosis of unexpected occurrences**

source: SPIE Vol 729 (Space Station II - 1986)

categories: Real Time Systems, Control, Fault Diagnosis

system: HERMIES-II *language:* PICON 10 *Pages*

institution: Oak Ridge National Laboratory *funded by:* DOE

WOLF87

author: Wolfe,A.

title: **An Easier Way to Build a Real-Time Expert System**

source: Electronics, March 5, 1987

categories: Real Time Systems, Control

system: G2 *language:* LISP 3 *Pages*

institution: GENSYM *funded by:*

WRIG86

author: Wright,M. Green,M. Fiegl,G. Cross,P.

title: **An Expert System for Real-Time Control**

source: IEEE Software, March 1986

categories: Real Time Systems, Planning

system: HEXSCON *language:* LISP, Pascal 9 *Pages*

institution: SRI *funded by:*

YEUN86

author: Yeung,D.

title: **Using Cnet on the iPSC**

source: DAI Group Research Note 20

categories: Distributed AI, Distributed Systems

system: MACE *language:* LISP 5 *Pages*

institution: USC *funded by:*

YONE77

author: Yonezawa,A. Hewitt,C.

title: **Modelling Disributed Systems**

source: IJCAI-77

categories: Distributed AI, Distributed Systems

system: ACTORS *language:* PLASMA 7 *Pages*

institution: MIT *funded by:* ONR

YUAN87

author: Yuan,X. Tripathi,S. Agrawala,A.

title: **Scheduling in Real-Time Distributed Systems - A Review**

source: NTIS

categories: Distributed AI, Distributed Systems

system: *language:* 37 *Pages*

institution: University of Maryland

funded by: Westinghouse

A.1 INTERLEAVING EXCEL FORMS IN WORD DOCUMENT

- **Word file printings**

Section 2.3.3 has all of the necessary extra pages already inserted into the document upon which all of the Excel and Draw files are to be reprinted. So the basically blank pages from this section are to be sorted out and run through the printer again to have the Excel forms printed on them. An example of how to sort and organize the pages for printing is listed in **Figure 1.**

- **Table file printings**

Each of the multiple page files in the Excel spreadsheets print in the same fashion. They start by printing with the upper left most page and then print the pages going down the spreadsheet first. And then Excel continues to print each column of pages as it moves across the spreadsheet to the right, until all of the pages are printed that have any data on them. Because of the way the spreadsheets are printed, a preprinted page containing the header, footer, and page number is generated by the Word document. These preprinted pages for the Excel forms need to be organized into a printing order, so that they can all be printed with one command rather than having to print each of the pages individually. **Section 2.3.3** has all of the necessary extra pages already inserted so that they are printed when that section is printed. These blank pages are to have the Excel forms printed on them. And they need to be retrieved from this section so that they can be printed on. An example of how to sort and organize the pages for printing is listed in **Figure 1.**

- **Chart file printings**

Printing the Charts are similar to the printing the Excel forms except that all of the charts are in a single folder for each table. All of the charts in a folder can be selected and printed at the same time. The charts will be printed in an alphabetical order. So the preprinted page numbers for the charts also need to be sorted into the order in which they will be printed. The major drawback about printing all of the charts at once is that all of the Word preprinted sheets would need to be loaded and someone could slip a printing request in between printing of the charts. This would use one of the preprinted sheets. So printing the charts would need to be done when there would be no interference, or chance of having someone else use the preprinted sheets that would be loaded. An example of how to sort the pages for printing is listed in **Figure 1.**

- **Appendix file printings**

Printing the appendix drawing example on a preprinted form is similar to the printing the Excel forms. So the preprinted page numbers for the appendix drawing needs to loaded for printing. The major drawback about printing on the Word preprinted sheets that need to be loaded is that someone could slip a printing request in before printing of the drawing and wind up printing on the preprinted sheet. So printing the files discussed here would need to be done when there would be no interference, or chance of having someone else use the preprinted sheets that would be loaded. An example of how to sort the pages for printing is listed in **Figure 1.**

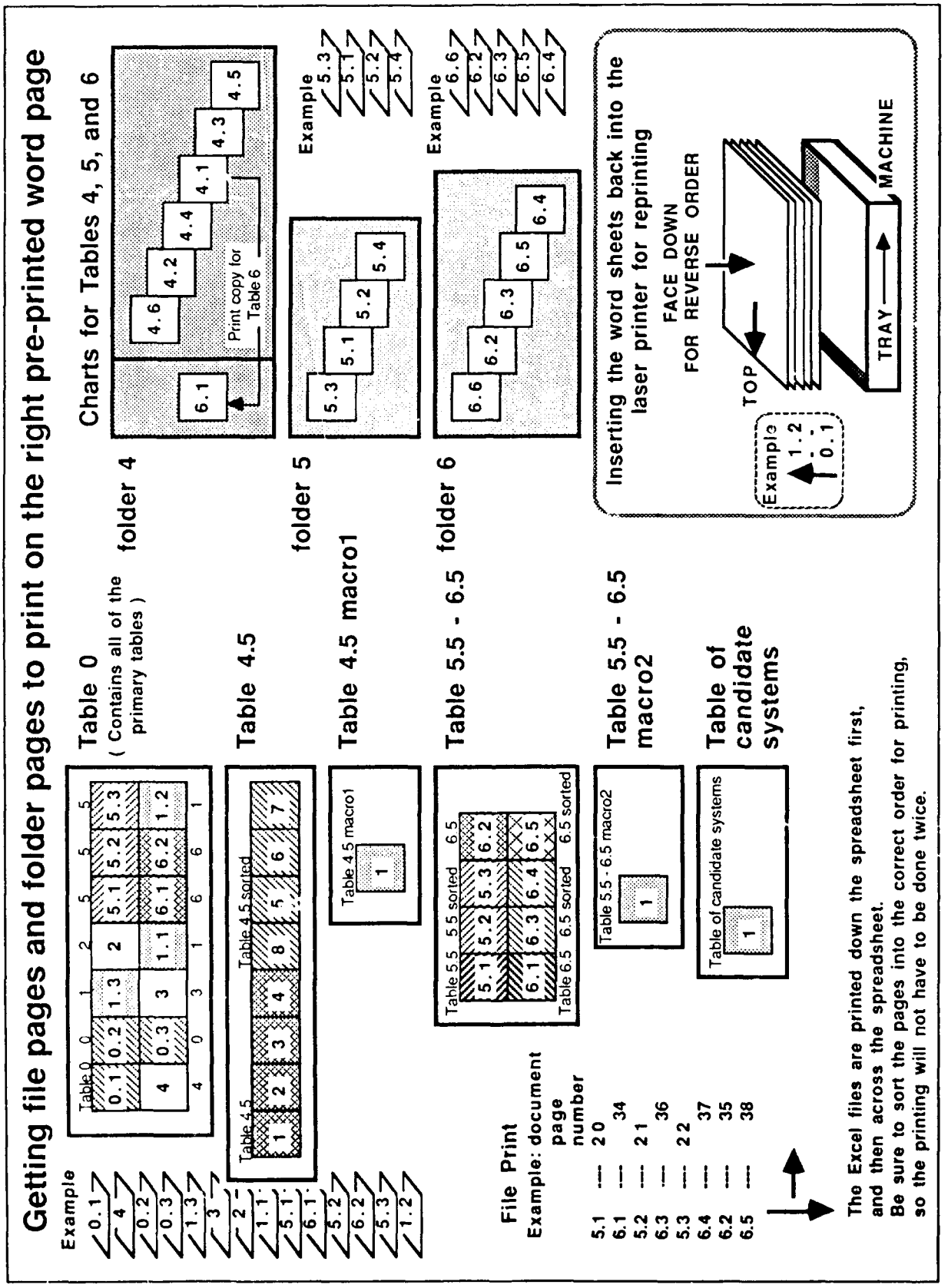


Figure A-1 - Printing Excel pages on a pre-printed Word form

☆U.S. GOVERNMENT PRINTING OFFICE: 1992 - 648-127/62189